RL-TR-91-22
Final Technical Report
March 1991

(2)

LITHIUM NIOBATE ARITHMETIC LOGIC UNIT

University of Colorado

Andrew R. Pleszkun

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

Rome Laboratory
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-91-22 has been reviewed and is approved for publication.

APPROVED: R J Michalak

RICHARD J. MICHALAK, Chief
Digital Optical Signal Processing Division
Photonics Laboratory

APPROVED: Donald W. Hanson

DONALD W. HANSON
Director of Surveillance & Photonics

FOR THE COMMANDER: B G Oaks

BILLY G. OAKS
Directorate of Plans & Programs

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL(OPA) Griffiss AFB, NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1991	Final Apr 89 - Aug 90

4. TITLE AND SUBTITLE
LITHIUM NIOBATE ARITHMETIC LOGIC UNIT

5. FUNDING NUMBERS
C - F30602-88-D-0026,
Task P-9-6009
PE - 62702F
PR - 4600
TA - P3
WU - P6

6. AUTHOR(S)
Andrew R. Pleszkun

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of Colorado
Dept of Electrical and Computer Engineering
Campus Box 425
Boulder CO 80309-0425

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Rome Laboratory (OCSP)
Griffiss AFB NY 13441-5700

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
RL-TR-91-22

11. SUPPLEMENTARY NOTES
Rome Laboratory Project Engineer: Joanne H. Maurice/OCSP/(315) 330-7670
Prime Contractor: Calspan-UB Research Center, PO Box 400, Buffalo NY 14225

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

Initial development of an optical-serial arithmetic unit is explored. The design is based on lithium-niobate directional-coupler devices as the optical switching elements. The devices can be interconnected by optical fiber to create logic functions with which digital-optic systems may be constructed. Circuit designs for implementing bit-serial multipliers and dividers are presented. Experimental procedures for characterization of lithium-niobate switch performance are described. Measurement of drive-electronics delay is addressed.

NOTE: Rome Laboratory/RL (formerly Rome Air Development Center/RADC)

14. SUBJECT TERMS	15. NUMBER OF PAGES
lithium-niobate directional-couplers, optoelectronic switching, bit-serial architecture, digital-optic signal processing, optical coupling	106
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

CONTENTS

Development of an Optical Serial Arithmetic Unit — 5 pages
Final Report

Attachment A
Design Alternatives for Optical Bit-Serial Multiplication
OCS Technical Report 89-37 — 41 pages

Attachment B
A Bit-Serial Optical Divider
Technical Report — 21 pages

Attachment C
Lithium Niobate Switch Evaluation
OCS Technical Report 90-16 — 14 pages

Development of an Optical Serial Arithmetic Unit
(Task No. P-9-6009 Delivery No. 0009)

Final Report

Andrew R. Pleszkun
University of Colorado - Boulder

1. Introduction

The goal of this project was to explore the initial stages in the development of an optical serial arithmetic unit. An additional goal was to transfer the expertise in the building of optical systems developed at the Optoelectronic Computing Systems (OCS) Center at the University of Colorado - Boulder to the Rome Air Development Center (RADC) in Rome, New York. The first goal was accomplished by the principal investigator with a graduate research assistant, working closely with members of the Digital Bit-Serial Optical Computing Project at OCS. The second goal was accomplished through the hiring of Dr. Robert Feuerstein to fill a post doctoral position. In this position, Dr. Feuerstein spent part of his time at RADC working the researchers in the Photonics Laboratory.

This Final Report for the project summarizes the accomplishments and activities during the contract period. More extensive details of the results of these activities can be found in the three technical reports that accompany this Final Report. These reports are titled: *Design Alternatives for Optical Bit-Serial Multiplication, A Bit-Serial Optical Divider* and *Lithium Niobate Switch Evaluation*. The first two reports describe our design activity while the third report is an outcome of Dr. Feuerstein's work at RADC.

2. Background for Arithmetic Unit Development

Our design activities and much of the work done at RADC revolve around the use of lithium niobate (LiNbO3) switches, also known as directional couplers. This device was selected as the basic element in our designs primarily for reasons of consistency and compatibility. The lithium niobate switch is the same device that used by the University of Colorado bit-serial optical computer, SCAMP [HeJP88]. Thus, the techniques developed for use of the switch at OCS may also be used for any digital optical projects at RADC.

The lithium niobate switch can be used as a five-port optical device in which ports A, B, and C are inputs and D and E are outputs. The operation of the device is simple. The switch can be placed in a bar state (the input at port A gets output at port E and port B is connected in a similar fashion to port D) or a cross state (A is connected to D and B is connected to E) depending upon the input at C. When the C input goes to a logical high value due to the presence of a light pulse, the switch is put into the bar state. Likewise, when no pulse is present at port C, the switch is placed into the cross state. Using such a device it is possible to create basic logic functions. Some of these logic functions and how they are implemented are summarized in [HeJP88].

In addition, all our designs are based on using several other basic circuits. These include the optical connector which has two inputs and a single output and performs the same task as a wired OR, a splitter that can be used to fan-out a signal to several places, a pulse stretcher that is used to elongate the duration of a light pulse, and finally the optical fiber. The optical fiber not only carries a signal from one place to another, but can also be used to delay the signal for a period of time. From these basic devices, one can easily construct a delay line memory and a single-bit full adder. The memory delay line depends on the ability of the optical fiber to delay pulses of light. This device circulates data pulses in an N time unit delay line.

While our design work revolved around the use of this device, a major portion of Dr. Feuerstein's time was spent in better characterizing the physical properties of this device. For example, issues such as crosstalk, loss and switching voltage are important parameters of the switch. A large circuit using such switches cannot be built unless these characteristics are well understood. An systematic approach for evaluating these switches is presented in [Feur90].

3. Multiplier and Divider Alternatives

Since the SCAMP project had already designed a machine with a simple ALU that performed addition, subtraction and other simple logic operations, we determined that the best place to start in designing a serial arithmetic unit was to investigate the design of a multiplier. In part this decision was made because of the expectation that the cost of the lithium niobate switches would limit the total number of devices used in any building of a prototype to something on the order of 30 switches. Since the switch budget was so limited, the thrust of our investigations regarding a multiplier unit involved looking at the tradeoff between the number of switches and the time needed to perform a multiply. The results of these designs are detailed in [DiPl89]. In that report, we have compared a number of multiplier designs in which the results of the comparisons are presented as functions of the wordsize (or length of the operands) on which the multiply is performed.

Not unexpectedly the design that uses the smallest number of switches (21) is a strictly serial multiplier using a single 1-bit adder. The time required to perform the multiplication of 2 N-bit numbers to produce a $2N$-bit result is $(N+1)^2$ units of time. However one design, that use 30 switches, can perform the same multiplication in $(N+2)(\frac{N}{2}+1)$ units of time, or in roughly half the time. This improvement in performance is achieved by arranging switches in the proper way so as to provide a limited amount of parallelism for a relatively small increase in the number of switches.

In these investigations, we also explored the possibility of using an **on-line** multiplication scheme. On-line schemes are geared towards processing a bit stream of information, and unlike the other bit-serial schemes, have the advantage of producing one bit of the quotient each clock tick (after some small initial delay). As expected these designs, have the disadvantage of requiring a rather large number of devices for their implementation In particular, depending on which of our 2 on-line designs are used, the device count is $6.5N$ or $\approx 2N + 4(N + \log_2 N)$, where is N is the number of bits in the word to be multiplied. More concretely, if we wish to multiply 2 32-bit numbers to produce a 64 bit results, so N=32, the number of optical switches needed using an on-line approach can be 208 or 221.

Once we were satisfied that we had explored the possibilities in the design of a multiplier, we changed our focus to the design of a divider unit. Division is much more difficult than multiplication in that there is little to no opportunity for parallelizing the operations. We are basically reduced to performing division by repeated subtractions, compares and shift operations. Thus, strictly serial division, like multiplication, requires $O(N^2)$ operations, where N is the length of the operand. The only opportunities for improving this performance is to make the repeated subtractions faster. Since subtractions are essentially the same operations as addition, techniques similar to those used in the multiplier design can be used. Details of these design options can be found in [DiPl90].

In our investigations, we found that once division could be performed, the square root operation could also be performed with 3 addition switches. Furthermore, due to the similarities between the subtraction operation in division and the addition operations in multiplication, a combined multiplier/divider/square rooter would require 31 switches.

Through the design process, all the circuits that were developed were also tested with a special design simulation tool called **Hatch**. This tool permits a designer to enter a design that uses optical switches and then simulates the circuit. The simulator accounts for loss in the switches and the optical fiber connecting switches. It also accounts for cross-talk within in the fiber. Hatch was written for the SCAMP project to run on a Macintosh computer. Copies of the program have been given to the researchers at RADC's Photonics Laboratory.

4. Interactions with RADC

Interactions with the Photonics Laboratory at RADC did proceed as smoothly as our design work primarily due to the difficulty in filling the post doctoral position. Although we placed an advertisement for the position and received resumes from approximately 70 people, most of these did not meet the needs of the position. In particular, the most pressing need was the ability to start work immediately, and secondly, the person had to be a U.S. citizen or at least have permanent residence status. In the middle of December, we interviewed Dr. Robert Feuerstein for the position and as a result of the interview offered him the position.

Before arriving in Boulder Dr. Feuerstein visited RADC during the third week of January. At the beginning of February, Dr. Feuerstein began work at OCS. During this month he familiarized himself with the technology used in the digital optical computer under construction at the OCS. He worked with the OCS staff on the drive electronics for the polarization dependent lithium niobate switches. As a result of this work, construction was begun on a new design for the switch drive electronics.

In March, 1990, Dr. Feuerstein visited the RADC Photonics Laboratory. During his stay he engaged in work on the digital optical arithmetic/logic unit. One of his first tasks was transferring the necessary fiber optic technology to the staff. With the RADC staff, ST type single-mode connectors were affixed to a number of lithium niobate switches, 3dB couplers and fibers. Procedures to evaluate the lithium niobate switch performance, using the equipment on hand, were developed. Dr. Feuerstein and Mr. Jim Rosetti discussed the drive electronics for the polarization independent switches. A draft of these procedures was prepared. At the request of the RADC staff, Dr. Feuerstein visited the factory site in Oakville, Ontario for training in the use of the Opto Electronics Inc.

Optical Time Domain Reflectometer (OTDR). After returning to RADC, he have a brief training session in the use of the OTDR to RADC staff. He also determined that the 1300nm optical source for RADC's OTDR was inoperative. Dr. Feuerstein also prepared and delivered a seminar on waveguide modes, coupled mode theory and the theory and operations of the LiNbO3 switches.

In April of 1990 Dr. Feuerstein worked at OCS on the drive electronics for the lithium niobate switches. The new electronics board was completed during this time and the 4-bit, bit serial digital optical counter was successfully operated.

In May of 1990, Dr. Feuerstein returned to RADC. Work continued on putting ST type single mode fiber optic connectors on lithium niobate switches, couplers and fiber jumpers. Evaluation of the switch properties; crosstalk, loss and switching voltage, continued. Refinement of the procedures used continued. A number of the switches were found to have optical losses larger than those specified by the manufacturer, Crystal Technologies Inc. (CTI)

A salesman from CTI visited RADC during this time and was consulted about the excess losses. He stated that these switches were fiber butt coupled at a time when the epoxy used was not temperature insensitive. Therefore, large losses could be explained by the fibers having moved relative to the waveguides in the lithium niobate substrate during temperature transients in shipping. He stated he would take back all switches which did not meet specified loss figures and replace them. It was determined that the switch testing should be a high priority so that all defective switches could be found and shipped back to CTI.

At the end of May, Dr. Feuerstein delivered a lecture entitled *Lithium Niobate Directional Coupler Switches in Digital Logic.* This lecture reviewed basic switch operation and how digital logic functions could be implemented using them. As an example of a small system, the design and operations of the bit-serial counter in operation at the University of Colorado was presented.

Following his return to Colorado, Dr. Feuerstein completed a technical report [Feur90] that describes a procedure to evaluate and characterize lithium niobate switches.

5. Conclusions

This contract has permitted the initial development of designs for an optical serial arithmetic unit. A wide variety of designs have been explored, as well as trade-offs in the switch count and performance of the designs. This can be seen in the attached technical reports. When the post-doctoral position was filled, interactions with RADC went smoothly. As much of the expertise found in OCS as possible was transferred to the Photonics Laboratory at RADC during the relatively short period of the contract. Based on Dr. Feuerstein work at the Photonics Laboratory, should the Photonics Laboratory be interested in further development of an optical serial arithmetic unit, dedicated technical support is needed to construct the electronic and optical components.

References

[DiPl89] M. P. Dickman and A. R. Pleszkun, "Design Alternative for Optical Bit-Serial Multiplication," *OCS Technical Report 89-37*, Dept. of Electrical and Comp. Eng., University of Colorado, Boulder, November 1989.

[DiPl90] M. P. Dickman and A. R. Pleszkun, "A Bit-Serial Optical Divider," *OCS Technical Report 90-22*, Dept. of Electrical and Comp. Eng., University of Colorado, Boulder, August 1990.

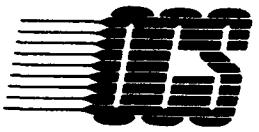
[Feur90] R. Feuerstein, "Lithium Niobate Switch Evaluation," *OCS Technical Report 90-16*, Dept. of Electrical and Comp. Eng., University of Colorado, Boulder, June 1990.

[HeJP88] V.P. Heuring, H.F. Jordan, and J.P. Pratt, "A Bit Serial Architecture for Optical Computing," *OCS Technical Report 8801*, Dept. of Electrical and Comp. Eng., University of Colorado, Boulder, Jan. 1988.

ATTACHMENT A

Design Alternatives for Optical
Bit-Serial Multiplication

Michael P. Dickman
Andrew R. Pleszkun



OPTOELECTRONIC COMPUTING SYSTEMS

AN

NSF

ENGINEERING RESEARCH CENTER

UNIVERSITY OF COLORADO
BOULDER, CO.

COLORADO STATE
UNIVERSITY
FORT COLLINS, CO.

Design Alternatives for Optical Bit-Serial Multiplication

Michael P. Dickman
Andrew R. Pleszkun

Optoelectronic Computing Systems Center
Campus Box 525
University of Colorado
Boulder, CO 80309-0525

OCS Technical Report 89-37

The Center is sponsored in part by the NSF/ERC grant number CDR 8622236 and by the Colorado Advanced Technology Institute (CATI), an agency of the State of Colorado.

Design Alternatives for Optical Bit-Serial Multiplication

1. Introduction

Since the early days of electronic computing, there has always been interest in high performance arithmetic functional units. In the first all electronic computing engines, such as the ENIAC, arithmetic operations were performed in a serial fashion. As electronic devices became cheaper to use, and in particular, the form of storage elements changed, the circuits that performed arithmetic operations became parallel in nature. Today, it is assumed that an arithmetic unit will perform arithmetic functions in a parallel fashion.

The development of optical switching devices has, however, necessitated the reexamination of design options memory organizations due to the cost of switching elements. This paper concentrates on possible designs for implementing a multiplier with optical switching elements. The basic switching elements are lithium-niobate switches interconnected by optical fibers. Since these devices are still relatively expensive, this paper looks at the tradeoff between the cost (number of switches) and time (in clock cycles) for performing a multiply operation.

In this paper, we first discuss basic multiplication, followed by a description of the basic optical devices and circuits that are used. Several multiplication circuits are then presented. These circuits vary in complexity from simple bit-serial multipliers to so-called on-line multipliers.

2. Basic Multiplication

Essentially all multipliers are based, in some way, on the classic shift and add multiplication algorithm. That is, all such devices generate shifted multiples of the multiplicand which are added to produce the final product. How this task is accomplished varies widely among different multipliers. Some multipliers utilize a highly parallel pipelined approach [Lyon76, ChWi79] whereas others are essentially sequential [Wake81]. Nonetheless, the basic multiplication

algorithm consists of adding shifted versions of the multiplicand in order to generate the final result. These shifted multiples of the multiplicand are determined by the digits of the multiplier. This is essentially how one would multiply two numbers by hand. For example:

```
   1 0  (multiplicand)
x 1 3   (multiplier)
-------
   3 0
 1 0
-------
 1 3 0
```

In binary, the shifted versions of the multiplicand are simply the multiplicand itself or zero. A typical binary multiplication might look as follows:

```
          1 0 1 0
      x   1 1 0 1
-------------------
          1 0 1 0
        0 0 0 0
      1 0 1 0
    1 0 1 0
-------------------
1 0 0 0 0 0 1 0
```

Instead of using a strictly binary representation of the numbers, some multiplier circuits use digit recoding and redundant number systems [Boot51, TrEr77, IrOw87] in order to quickly and efficiently multiply two numbers. Because of the simplicity of using binary numbers in multiplication and the fact that all input and output to these multipliers will be in binary, the following discussion only considers multiplication using a binary number representation.

3. Basic Devices and Circuits

For the work presented here, only existing optical devices will be considered for implementation of parallel and sequential multiplication algorithms. For reasons of consistency, the devices that will be used are the same as those used by the University of Colorado's bit-serial optical computer, the SCAMP [HeJP88]. (Of course the actual physical characteristics of the devices may vary somewhat depending on the manufacturer). The primary device that is used is the lithium-niobate switch (see Figure 1). This switch can be used as a five-port optical device in

which ports A, B, and C are inputs and D and E are outputs. The operation of the device is simple. The switch can be placed in a bar state (the input at port A gets output at port E and port B is connected in a similar fashion to port D) or a cross state (A is connected to D and B is connected to E) depending upon the input at C. When the C input goes to a logical high value due to the presence of a light pulse, the switch is put into the bar state. Likewise, when no pulse is present at port C, the switch is placed into the cross state.

Using such a device it is possible to create several logic functions. For example, an AND function can be realized by equating the two AND inputs to the C and A terminals of the switch and obtaining the output from the E terminal of the switch. Notice that the B input is not used for the AND function. In a similar manner, the Exclusive-OR function $Z = (X \cdot \bar{Y}) + (\bar{X} \cdot Y) = X \oplus Y$, can be implemented by equating the X input to the C terminal, the Y input to terminal A, and its complement, $\bar{Y}$, to the B terminal. The output, Z, is taken from the D terminal.

Another basic device is the optical connector (see Figure 2). This device has two inputs and a single output and performs the same task as a wired OR. A third device consists of a splitter (see Figure 3). The splitter can be used to fan-out a signal to several places. The pulse stretcher (see Figure 4), is used to elongate the duration of a light pulse. This device is useful when the optical switch must be kept in the bar state for longer periods of time. The final device is the optical fiber (see Figure 5). The optical fiber not only carries a signal from one place to another, but can also be used to delay the signal for a period of time.

Two simple, but important circuits, implemented with these optical devices include the memory delay line [Sara89] and the single-bit full adder. These circuits are displayed in Figures 6 and 7 in a simplified format such that the splitters, connectors, and optical fibers are not explicitly shown. Figures in Appendix A display these circuits in a format that explicitly shows all optical devices.

The memory delay line depends on the ability of the optical fiber to delay pulses of light. This device circulates data pulses in an N time unit delay line. A data pulse can be written into the loop by asserting the write input high and placing the data on the data line. The data then travels through the fiber delay line and emerges N time units later. The data can then be read at the output terminal of the circuit. The output of this circuit is then fed back into the delay line, thus storing the data in a continuous loop. The second switch (the one connected to the clock) is simply a regenerator for the data pulses. The clock generates a constant stream of pulses. When a high signal comes out of the delay line and into the C terminal of this switch, the switch is placed in the bar state and the clock's high pulse gets transmitted to the output. This regeneration process is necessary because a signal cannot circulate endlessly in the delay line loop without substantial attenuation.

Unlike the delay line memory, the implementation of the full adder does not depend upon delay due to the optical fiber. Instead, The adder is an example of using switches to implement a logic function. The inputs to the adder include the two bits to be added, X_{in} and Y_{in}, the complement of X_{in}, $\bar{X}_{in}$, and the carry in bit, C_{in}. By simply applying the proper logic function to the inputs via switches and combiners, the proper full adder outputs are generated. One can easily verify that the adder circuit does, in fact, implement the following two logic equations: $S_{out} = X_{in} \oplus C_{in} \oplus Y_{in}$ and $C_{out} = (X_{in} \cdot Y_{in}) + (X_{in} \cdot C_{in}) + (Y_{in} \cdot C_{in})$. The same techniques as those used to create a memory delay line and a full adder can be applied to the design of multiplier circuits.

4. Multiplier Circuits

Any device which performs multiplication must accomplish two tasks, the generation of the shifted multiplicands and the addition of these values. The first task can be completed by simply AND-ing the bits of the multiplicand with the appropriate bit of the multiplier. Adding the numbers together, however, is not quite as simple.

In the devices we will be using, binary numbers will be stored as continuously moving pulses of light in an optical fiber. This means that numbers must be stored and accessed bit serially, least significant bit to most significant bit or vice versa. Addition of two numbers must, therefore, be done one bit at a time. Adding two n-bit numbers requires n single-bit additions. Similarly, multiplication of two n-bit numbers requires the addition of n n-bit numbers, for a total of, $n(n-1)$ single-bit additions. Thus, multiplication is $O(n^2)$ with respect to single-bit addition. One would, therefore, expect to see a multiplier with a single adder take at least n^2-n cycles to complete the multiplication of two n-bit numbers. Likewise, one might expect a device which multiplies the same two numbers but has two adders to take $\frac{n^2}{2}$ cycles to complete the multiplication.

4.1. The Serial Adder

One of the simplest multipliers to design uses a single one-bit full adder. The difficult part of such a design is to get the proper bits into the adder at the appropriate time for all n^2 single-bit additions. To accomplish this, four separate memory delay lines are used. The delay lines store the multiplicand, the multiplier, the result, and the least significant bit of the multiplier. Manipulation of these delay lines to perform a multiplication is simple. One fills the multiplicand and multiplier delay lines with the binary numbers to be multiplied while the result delay line is being cleared. The multiplicand is then ANDed with the least significant bit of the multiplier. The ANDed multiplicand is added to the result (which is initially zero) to produce the first partial product. The result and multiplier are then shifted right one bit position. Any carry out of the previous addition iteration is shifted into the most significant bit of the result and the least significant bit of the result is shifted into the multiplier loop. In this manner, the bits of the multiplier that have already been used are replaced with the low order bits of the result. The next least significant bit of the multiplier is then accessed and the ANDing and addition process is repeated. After n iterations of ANDing, adding and shifting, the final product of the multiplication is

produced. The least significant bits of the result are located in the multiplier loop and the most significant bits are located in the result loop.

A block diagram displaying the implementation of this multiplier is shown in Figure 8. For simplicity, the delay line and adder of Figures 6 and 7 are represented as boxes. The feedback loop of the delay line memory, however, is not included in its block representation. The multiplier and multiplicand are read into their respective delay lines and the result delay line is cleared when the start signal is held high. After N clock cycles, the first bit of the multiplier will emerge and be clocked into the least significant bit delay line by the $N+1$ clock. During the next cycle, the first bit of the multiplicand is produced. This bit is ANDed with the least significant bit by switch #1 and is then added to the result in the full adder. The carry into this first bit must be clear. This is accomplished by switch #2. The result of the addition is then placed back into the result delay line, replacing the previous partial product result bit. These additions continue until all N bits of the multiplicand have been added to the result. When these single bit additions have finished, the carry out is shifted into the result delay line by using switch #3, the least significant bit of the result is shifted into the multiplier loop by switch #4, and the next least significant bit of the multiplier is clocked into the least significant bit delay line. The shifting of the result and multiplier also occur during this cycle. The multiplicand, since it is stored in a delay line that is one time unit longer than the result and multiplier, is not shifted. Instead, the multiplicand simply reaches its original position during this cycle. The multiplication completes after N iterations through this shift and add loop, with the product appearing at the result outputs.

As expected, this design takes $O(n^2)$ to complete its operation and requires 21 optical switches. The actual time is $(n+1)^2$ because of the time it takes to load the numbers into the circuit. The multiplier has been fully designed and tested with Hatch, a digital optics design tool. The complete multiplier circuit and timing generation circuit, which produces the $N+1$ clock, are located in Appendix A. As shown in the Appendix, the control circuitry generates a busy signal. This signal is provided to surrounding support circuitry and indicates when the multiplier is

finished with its operation. If such a signal is not needed, the multiplier can be implemented with 18 switches instead of 21.

4.2. Using Two Full Adders

By using two adders, one can essentially cut the time required for a multiply in half. There are several ways in which one can effectively perform the addition of two bits in parallel. One might simultaneously add two bits of the shifted multiplicand with two bits of the result (partial product). Another strategy is to add two shifted/ANDed versions of the multiplicand into the result at the same time. This second approach can be thought of as multiplying the multiplicand by two bits of the multiplier simultaneously. This can also be thought of as recoding the multiplier bits into the digits 0, 1, 2, or 3.

Based on the first strategy, one can take two sequential bits of the shifted multiplicand and add them to two sequential bits of the partial product. This can be done using a simple two-bit full adder and leads to a design we call the odd/even multiplier. Unfortunately, each delay line can produce only a single-bit per cycle. Waiting two cycles to get two bits and then adding them together is no better than using a single adder which adds a bit every cycle. Instead, one needs to double the number of delay lines in such a way that two bits can be delivered to the two-bit adder every cycle. Using the addition of two sequential bits every cycle implies that the odd bits of a number need to be stored in one delay line while the even bits are stored in a completely separate delay line loop. This approach requires that the multiplicand and the result also be stored in two delay lines each.

Furthermore, and not quite as obvious, is the fact that the multiplier must also be stored in two delay lines. Even though only one bit is read from or stored into this delay line every iteration, each iteration length is half as long. This implies that one needs to read the current least significant bit every $\frac{n}{2}$ cycles which forces the length of the multiplicand memory delay line to be $\frac{n}{2}$. A delay line with a length of $\frac{n}{2}$ delays can only hold $\frac{n}{2}$ bits, thus, two such delay lines

are required to hold the n-bit multiplier. The other parts of this type of multiplication scheme are essentially the same as in the single-bit multiplier with the obvious exception that the single-bit full adder is replaced with a two-bit full adder.

Figure 9 contains a block diagram of the odd/even multiplier. This circuit functions essentially the same way as the single adder multiplier. The multiplicand and multiplier are input, in odd/even format, by holding start high. Switches #1 and #2 then AND the current least significant bit of the multipler with the multiplicand. The result of this ANDing is then added to the result (partial product) using the two-bit adder. This adder accepts a single carry and two sets of addends to produce a carry out and two sum bits. This adder consists of two single-bit full adders in which the carry out of the low addition is rippled into the carry in of the high addition. The result bit or the carry is then shifted into the result delay line depending upon switch #4. Switch #5 is used to shift the least significant bit of the result into the multiplier. The shifting of the result and multiplicand are accomplished by placing the current odd bits into the even bit delay line and vice versa. Thus, bit 2 can become bit 1 and bit 3 can become bit 2 and so forth. Furthermore, the odd bit delay lines have a time delay length that is one shorter than the other delay lines. This enables the least significant bit (bit 0), to be shifted out of the result or multiplier and the most significant bit (bit $N-1$) to be shifted in. After N iterations the product emerges at the result and multiplier delay lines in odd/even format. The complete odd/even multiplier consists of 32 switches and completes a multiplication of two n-bit numbers in $(\frac{n}{2}+1)(n+1)$ cycles.

This two-bit design, however, has one major drawback. Any multiplication device would most likely receive the multiplier and multiplicand bits sequentially. That is, these numbers would arrive with an odd bit followed by an even bit followed by an odd bit and so on. One must then split these even and odd bits and store them into their respective delay lines. Unfortunately, the even bits (and the odd bits) are produced every other cycle. Therefore, storing the even and

odd bits directly into their respective delay lines would result in gaps between every bit.

One possible solution to this problem might be to delay each bit by a different amount of time so that the bits will arrive at their destination one directly after another. This implies that the first odd or even bit must be delayed one unit longer than the second bit which is delayed one unit longer than the third bit and so forth. Figure 10 shows a design that implements this approach. The bits are delayed by the optical fibers and separated by the switches. The bits are then combined in the proper odd/even format. Such a solution requires the use of N extra switches for each number that must be divided into its odd and even bits.

Another solution to this problem involves continually looping the multiplicand and multiplier input bits until the odd and even bits can be written into their respective delay lines at the proper location with respect to the other bits (see Figure 11). The number to be split among its odd and even bits is stored in the delay line loop feed by the input signal. Whenever the output of this loop coincides with the proper timing for the input to the odd and even loops, the control signal goes high, thus, clocking the bit into the odd or even loop. It is important to note that the odd and even bit destination delay loops must also loop around until all the bits have been written into them. Such a solution, although only using six extra switches, requires $\frac{n^2}{2}$ time cycles.

The need to split the bits into odd and even parts renders an odd/even multiplier impractical. The splitting process requires switch count that grows with the size of the number being split or takes too much time when a constant number of switches are used. Such a multiplier might be reasonable, however, if numbers can be stored in this odd/even fashion without the need for a conversion for every multiplication.

4.3. Upper/Lower Multiplication

An alternative approach, but one that still performs two single-bit additions per cycle, can eliminate the problem of splitting up the odd and even bits by dividing the multiplicand and

result between their high $\frac{n}{2}$ bits and their low $\frac{n}{2}$ bits. Splitting a number between its high and low parts is a simple operation which actually requires no extra switches. This operation is done by simply gating the multiplicand and multiplier into delay lines at the appropriate time. The addition of the low $\frac{n}{2}$ bits is then carried out in parallel with the addition of the high $\frac{n}{2}$ bits on two separate single-bit full adders. Adding two n-bit numbers and multiplying two n-bit numbers would take the same amount of time as the previous multiplier, namely $\frac{n}{2}$ and $\frac{n(n-1)}{2}$ time cycles respectively.

Such an approach, however, has its own unique pitfalls. Every iteration of additions may result in a carry coming out of both the high and low $\frac{n}{2}$-bit additions. The carry coming out of the high order addition can simply be shifted into the most significant bit of the result in the same way as the previous multiplier designs. The carry out of the low order addition, however, must be added into the least significant bit of the current high order addition. Since the high and low bit additions are being done in parallel, this is not directly possible. One might, however, choose to add this carry into the most significant bit of the low order addition during the next iteration. This works because the high order least significant bit gets shifted into the most significant bit of the low $\frac{n}{2}$ bits of the result after every iteration. One must also ensure that this low order carry is added into the final result after the last iteration has completed. Such a high/low multiplier implementation requires 51 switches.

A different method of adding the low order carry bits into the result can reduce the number of switches from 51 to 44. Instead of adding in the low order carries on the very next iteration, all of the low order carries can be saved until the multiplication finishes. These can then be added as a final step to generate the result. This multiplication scheme again uses two single-bit full adders, but one of these has its inputs multiplexed in order to achieve the final addition of the stored carries into the result. Such a device also requires two extra delay lines which store the

carries out of the low order addition.

The implementation of this carry save high/low multiplier is shown in Figure 12. The multiplicand and multiplier are input in their usual bit serial representations. Both numbers are delayed so that the high bits and the low bits arrive at their respective delay lines at the same time. The bits are then clocked in by the start signal. Switches #1 and #3 AND the multiplicand to produce the appropriate version required for the addition. These ANDed numbers are then added to the result, producing a sum which is stored back into the result delay lines. The carry out of the low order addition is clocked into the carry delay lines by the $\frac{N}{2}+2$ clock. Switches #10, #11, and #12 are used to shift the result and multiplier during the shift cycle. After N iterations of the shifting and adding, the partial products (which do not include the low order carries) are present at the outputs of the result and multiplier delay lines. Switches #10 and #11 and the low order carry delay lines are then set up so that the high order bits will shift directly into the low order delay lines. This enables one to access all N bits of the result, multiplier, and stored carry sequentially from the low order delay line output. The second full adder is then multiplexed with switches #7, #8, and #9 so that the stored carries can be added to the partial product to produce the final result. The N-bit result and multiplier are combined to form the $2N$-bit partial product through switch #6. This $2N$-bit value is then added with the proper shifted version of the stored carry to produce the final product. An extra switch (that is not shown) is required to zero all the inputs into the adder from the carry loop except the actual N bits of the stored carry. This multiplier has been designed and simulated on Hatch. The actual circuit is shown in Appendix B. Both versions of the high/low two-bit adder require $n(\frac{n}{2}+2)$ time cycles to complete the multiplication of two n-bit numbers.

4.4. Simultaneously Using Two Bits of the Multiplier

A different approach to adding two of the n^2 bits together in parallel involves simultaneously adding two rows of the shifted multiplicand into the result. This does not use two unique adders, but instead uses an adder that is capable of adding 0, 1, 2, or 3 times one input (in this case, the multiplicand) together with the other input. By using such a device, one can essentially multiply the multiplicand by two bits of the multiplier at a time. Thus, only $\frac{n}{2}$ iterations of the n-bit additions are required. The expected time to multiply two n-bit numbers using such an algorithm is $n(\frac{n}{2} - 1)$.

The special two-bit adder required for this type of multiplication is shown in Figure 13. The inputs include the two inputs to be added, X_{in} and Y_{in}, two carry inputs, C_{in} and $C+1_{in}$, and the two inputs which indicate what multiple of the first addend to add to the second addend, LSB and LSB+1. The outputs from the adder include a sum and two carries. In a typical full adder the sum bit has a weight of one and the carry has a weight of two. In this adder, the sum has a weight of one, the first carry a weight of two, and the second carry has a weight of four. The inputs to the adder can sum up to a maximum of seven (the first addend times three plus the second addend plus a carry with the weight of one and a carry with the weight of two). The three-bit output is, therefore, sufficient to represent all possible inputs. The adder itself is a straightforward logical implementation requiring ten optical switches.

The implementation of this multiplier resembles that of the original single-bit multiplier. Figure 14 shows a block diagram for this circuit. Since only one bit of the multiplicand, result, and multiplier are required in a single cycle, they can be stored in a single loop delay line. The fact that the two least significant bits of the multiplier are required for each iteration of the n-bit additions requires the use of two delay lines to store these bits. Furthermore, since this device multiplies the multiplicand by two bits at a time, one needs to shift both the multiplier and the result by two bits after each iteration. This is accomplished by using delay lines for the multiplier

and result that are two time units less than the multiplicand. The carries out of the adder are shifted into the most significant bit of the result during this two bit shift, the low order carry is shifted in first followed by the high order carry. This multiplier requires only 30 switches and completes a n-bit by n-bit multiplication in $(n+2)(\frac{n}{2}+1)$ time cycles. The multiplier has been simulated on Hatch. The complete circuit diagram of this two-bit multiplier is included in Appendix C.

5. On-Line Multiplication

All of the multipliers considered so far have been sequential. That is, the shifted versions of the multiplicand are added one at a time. One major advantage of these multipliers is that they are essentially independent of the word size. Multiplying numbers of different bit lengths simply requires changing the delay in some of the optical fibers. These multipliers, however, require a certain amount of overhead in the form of delay lines to store values such as the multiplicand and multiplier. Furthermore, as seen in the appendices, some logic is required to control these multipliers. One can use a different approach to multiplying numbers by taking a parallel pipelined approach. Such multipliers produce results on-line, meaning that after a small delay from receiving the first input bits, the first output bit emerges. Each successive output bit then emerges one cycle after the other. Such multiplication schemes do not require any delay line storage or control pulses. Unfortunately, the number of switches needed to implement such multiplier circuits varies with the number of bits in the operands and can be quite large for large word sizes.

The basic on-line multiplier simultaneously takes all n shifted versions of the multiplicand and adds them together using $n-1$ full adders. The shifted versions are produced simply by ANDing several copies of the multiplicand with the bits of the multiplier. One can think of this as a serial version of a Wallace Tree. Figure 15 shows a circuit that generates shifted versions of the multiplicand. These versions are delayed by the proper time interval to accomplish the shifting. The shifted versions of the multiplicand are then added together using some interconnection

of $n-1$ adders.

Figure 16 shows two possible connections of adders for a n-bit by 4-bit multiply. This diagram does not indicate that each adder requires the complement of one of its three inputs. Since each adder can naturally produce the complement of its output, only the initial adders in each adder interconnection scheme requires one of its inputs to be complemented. Since the cost of inverting a bit is one switch, the first (tree) interconnection scheme requires $\frac{n}{2}$ additional complementing switches and the second (pipeline) scheme requires one extra inverting switch. This multiplication circuit requires (6.5n-4) switches for the tree interconnection of adders and 6n-3 switches for the pipeline interconnection.

Another possible implementation of an on-line multiplier is called the counting columns method and involves simultaneously adding all the bits in each column to produce a sum and carries to the next columns [Swar73b, Dadd83]. For example, consider:

```
        1 1 1
  x     1 1 1
-------------
        1 1 1
      1 1 1     sum bits
    1 1 1
1 1 0 1 0 0 2's place carry
0 1 0 0 0 0 4's place carry
-------------
1 1 0 0 0 1
```

In the counting columns approach, the right most column is added first, just as in manual multiplication. In this example, there is a single one including the carries in this column so the sum is one and the carries out are zero. One then looks at the next column which contains two ones. The sum bit of this column is zero and the 2's place carry (which gets added into the next column) is one. The third column contains four ones. This produces a zero sum bit a zero 2's carry and a one 4's carry. This carry goes two columns to the left because it is a 4's place carry. One continues adding the columns in the same fashion until the multiplication is completed.

The implementation of this algorithm, for a n-bit by 4-bit multiplication, requires the generation of the column bits at the correct time and a counter [Swar73a, Dadd80] which outputs the count of the inputs as a binary output. As it turns out, the column bits are generated by the same circuitry that generates the inputs into the previous parallel multiplier (see Figure 14). An n-bit by 4-bit multiplication requires a six input counter, four inputs for the sum bits and two inputs for the carries. A 4-bit by 4-bit multiply, however, can be accomplished by using a five input counter. Since the two's carry and the first number digit can never simultaneously be ones, their inputs can be combined into a single counter input. This reduces the number of counter inputs from six to five. The design of such a circuit is shown in Figure 17.

The implementation of the n-bit counter required for this type of multiplier can be accomplished using combinations of half and full adders. All of the input bits to a counter have a common weight of one. By using $\lceil \frac{N}{3} \rceil$ full adders, these N bits with weight one reduce to $\lceil \frac{N}{3} \rceil$ bits with a weight of two and $(\lceil \frac{N}{3} \rceil + N \bmod 3)$ bits with a weight of one. The bits with a weight of one can be again reduced into bits with weights of one and two by using another level of full adders. The bits of weight one are repeatedly reduced in this way until only one or two remain. If two bits of weight one remain, they can be reduced in one more step using a half adder. The remaining bit with a weight of one is interpreted as the one's place output of the counter.

The reduction method used to produce a bit of weight one and bits of weight two from bits of weight one can also be used to produce a bit of weight two and bits of weight four from bits of weight two. This method is repeated until all that remain are single bits that each have a different weight. A design for such a five input counter is shown in Figure 18. This diagram does not show that the complement for one of the inputs to each adder must be made available. Since each adder can produce the complement of its sum output at no extra switch cost, this 5-bit counter requires two extra complementing switches. Because a full adder can be implemented with four switches, and a half adder requires two, this 5-bit counter requires 12 switches. The number of

switches needed to implement a counter multiplier consists of the number of switches required to implement the counter plus the number of switches required to produce the column bits ($2N$). Thus, a n-bit by 4-bit counter multiplier requires 20 switches, or 5 fewer switches than the implementation of the previously described on-line multiplier.

Unfortunately, the switch count advantage of the counter multipler over the pipelined parallel multiplier disappears for larger values of N. This can be attributed to the fact that, for large values of m, the number of adders required in an m-bit counter approaches m. Building a counter requires $\frac{m}{3}$ initial adders to add the m input bits. These adders produce $\frac{m}{3}$ bits of weight one and $\frac{m}{3}$ bits of weight two. The $\frac{m}{3}$ bits of weight one are then further reduced with $\frac{m}{9}$ adders. This reduction results in $\frac{m}{9}$ bits of weight one and increases the number of bits of weight two to $\frac{m}{3} + \frac{m}{9}$. As this process is repeated, the number of adders required to reduce m bits of weight one into one bit of weight one is $\frac{m}{3} + \frac{m}{9} + \frac{m}{27} + \ldots \approx \frac{m}{3} \cdot \left[\frac{1}{1 - \frac{1}{3}} \right] = \frac{m}{2}$ adders for large values of m. This reduction step also produces $\frac{m}{2}$ bits of weight two. Using this reasoning, one can determine that for large m, it takes approximately $\frac{m/2}{2} = \frac{m}{4}$ adders to reduce the $\frac{m}{2}$ bits of weight two. One must then use $\frac{m}{8}$ adders to reduce the digits of weight four. When m is large, the total number of adders required to count m bits is $\frac{m}{2} + \frac{m}{4} + \frac{m}{8} + \ldots \approx \frac{m}{2} \cdot \left[\frac{1}{1 - \frac{1}{2}} \right] = m$ adders.

Thus, assuming that an adder requires at least four switches to build (some may require complementing switches), one might expect the implementation of an m-bit counter to require approximately $4m$ switches. The counting columns method of on-line multiplication, requires a $(N + \log_2 N)$- bit counter to multiply two N-bit numbers. The first term, N, comes directly from the N bits in each column and the $\log_2 N$-bits represent the carries which must also be added into

each column. An N-bit counter multiplier, therefore, requires approximately $2N+4(N+\log_2 N)$ switches (recall the $2N$ switches are required to generate the column bits).

6. Summary and Conclusions

The design of several types of multipliers using optical switches has been discussed. The times and switch complexity of each type of multiplier are summarized in Table 1. As might be expected, a large difference exists between the parallel and sequential multipliers. The parallel multipliers are capable of producing results on-line. Unfortunately, the complexity of these circuits grows linearly with the size of the numbers to be multiplied. On the other hand, the sequential circuits offer a switch count that does not vary with the size of the multiplicand and multiplier. However, these circuits require $O(n^2)$ time to complete a multiply. By using somewhat more hardware, the sequential switch can be made more parallel with the expected improvement in performance. To get a better feel for the relative switch counts and multiply times, Table 2 shows these values for multiply circuits using the various schemes on 16 bit and 32 bit inputs. As can be seen from Table 2, for the on-line multipliers, the switch counts grow quite large for even 16 bit inputs. The odd/even approach, using two adders, looks attractive until the delay or switches required to perform conversions are included. The high/low multipliers are more attractive since at the cost of a few clock cycles, multiplying by two bits at a time can save between 14 and 21 switches. Finally, the serial multiplier using a single adder, not unexpectedly, uses the smallest number of switches and the expected number of cycles to perform a multiplication.

References

[Boot51] A.D. Booth, "A Signed Binary Multiplication Technique," *Quarterly Journal of Mechanics and Applied Mathematics*, Vol. IV Part 2, 1951.

[ChWi79] I. Chen, R. Willoner, "An 0(n) Parallel Multiplier with Bit-Sequential Input and Output," *IEEE Trans. Computers*, Vol. C-28, No. 10, October 1979, pp. 721-727.

[Dadd83] L. Dadda, "Some Schemes for Fast Serial Input Multipliers," *IEEE Proceedings 6th Symposium Computer Arith.*, June 1983, pp. 52-59.

[Dadd80] L. Dadda, "Composite Parallel Counters," *IEEE Trans. Computers*, Vol. C-29, No. 10, October 1980, pp. 942-946.

[[HEJP88] V.P. Heuring, H.F. Jordan, and J.P. Pratt, "A Bit Serial Architecture for Optical Computing," *OCS Technical Report 8801*, Dept. of Electrical and Comp. Eng., University of Colorado, Boulder, Jan. 1988.

[IrOw87] M.J. Irwin and R.M. Owens, "Digit-Pipelined Arithmetic as Illustrated By the Paste-Up System:" A Tutorial," *Computer*, April 1987, pp 61-72.

[Lyon76] R.F. Lyon, "Two's Complement Pipeline Multipliers," *IEEE Trans. Commun.*, Vol. COM-24, April 1976, pp. 418-425.

[Sara89] D.B. Sarazin, "Fiber-Optic Delay Line Storage in Digital Optical Computers Based" on Directional Couplers," *OCS Technical Report 89-04*, Department of Electrical and Computer Engineering, University of Colorado, Boulder, Jan. 1989.

[Swar73a] E.E. Swartzlander, Jr., "Parallel Counters," *IEEE Trans. Computers*, Vol. C-22, No. 11, November 1973, pp. 1021-1024.

[Swar73b] E.E. Swartzlander, Jr., "The Quasi-Serial Multiplier," *IEEE Trans. Computers*, Vol. C-22, No. 4, April 1973, pp. 317-321.

[TrEr77] K.S. Trivedi and M.D. Ercegovac, "On-Line Algorithms for Division and Multiplication," *IEEE Trans. Computers*, Vol. C-26, No. 7, July 1977, pp. 681-687.

[Wake81] John F. Wakerly, "Miocrocomputer Architecture and Programming," John Wiley and Sons, 1981, pp. 98-101.

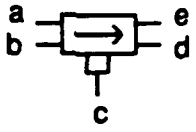


Figure 1. The lithium-niobate switch.

Figure 2. The optical connector.

Figure 3. The optical splitter.

Figure 4. The pulse stretcher.

Figure 5. Optical Fiber.

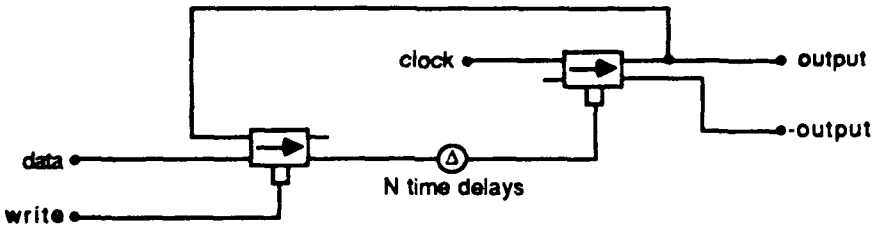


Figure 6. Delay Line Memory.

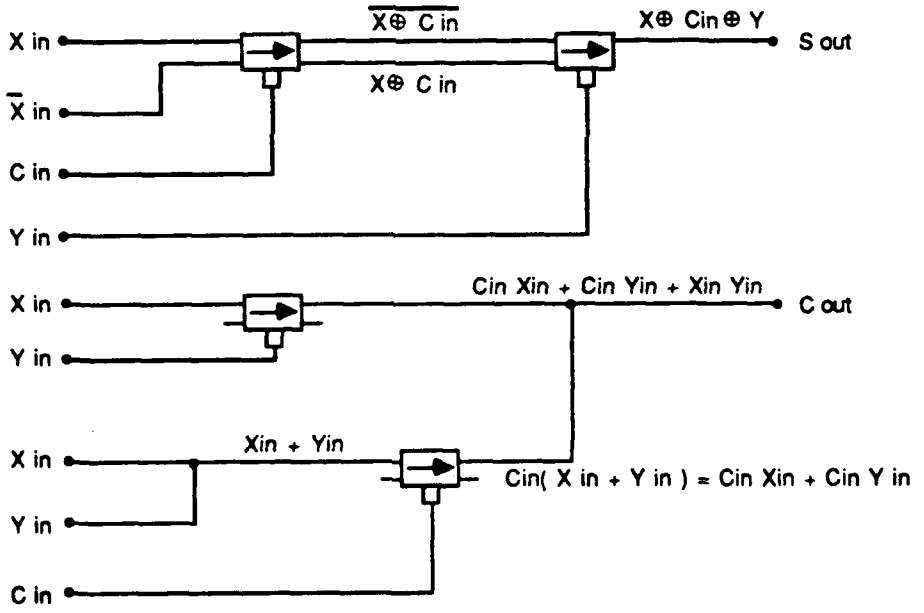


Figure 7. Single Bit Full Adder.

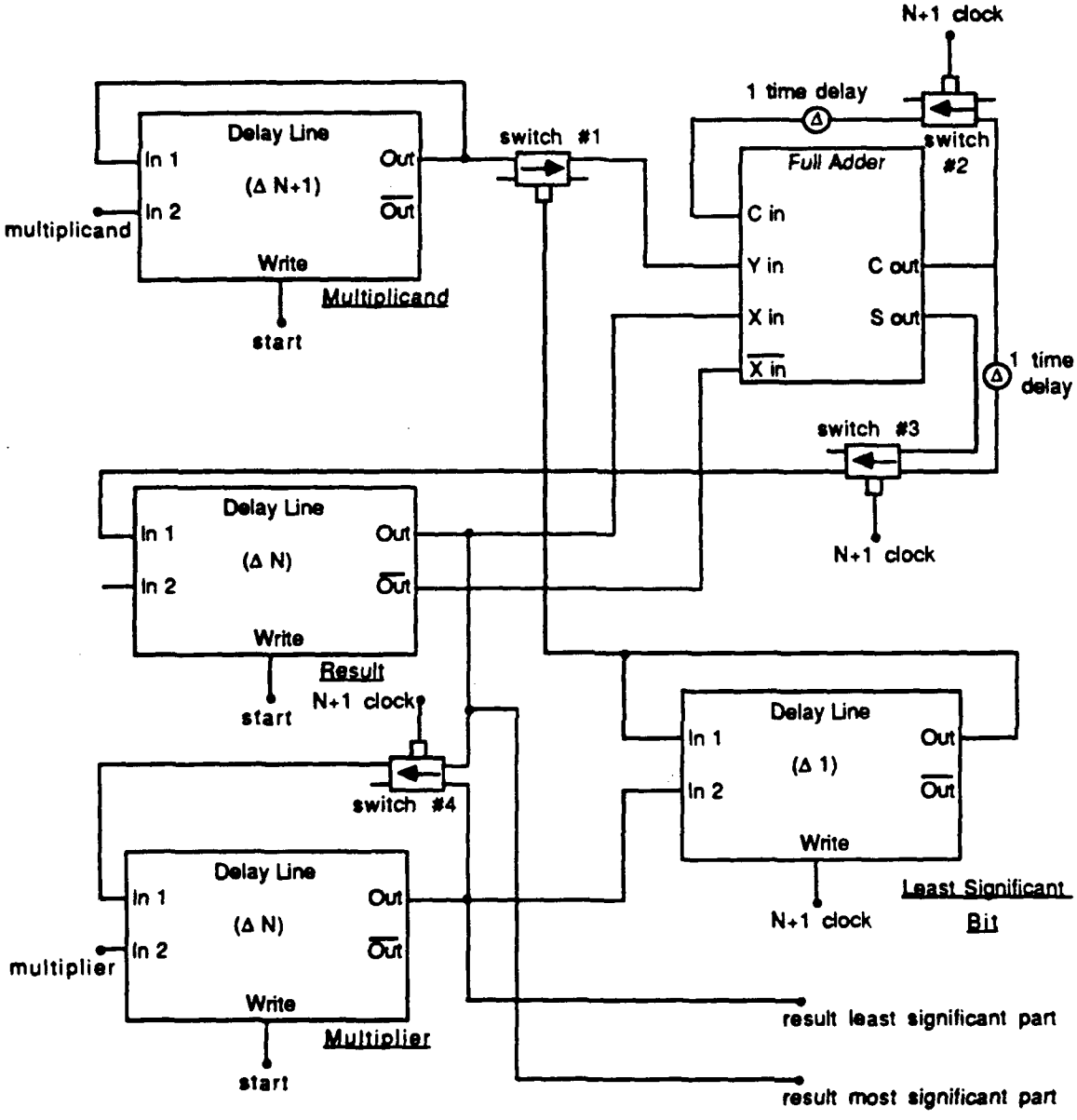


Figure 8. Single Bit Serial Multiplier.

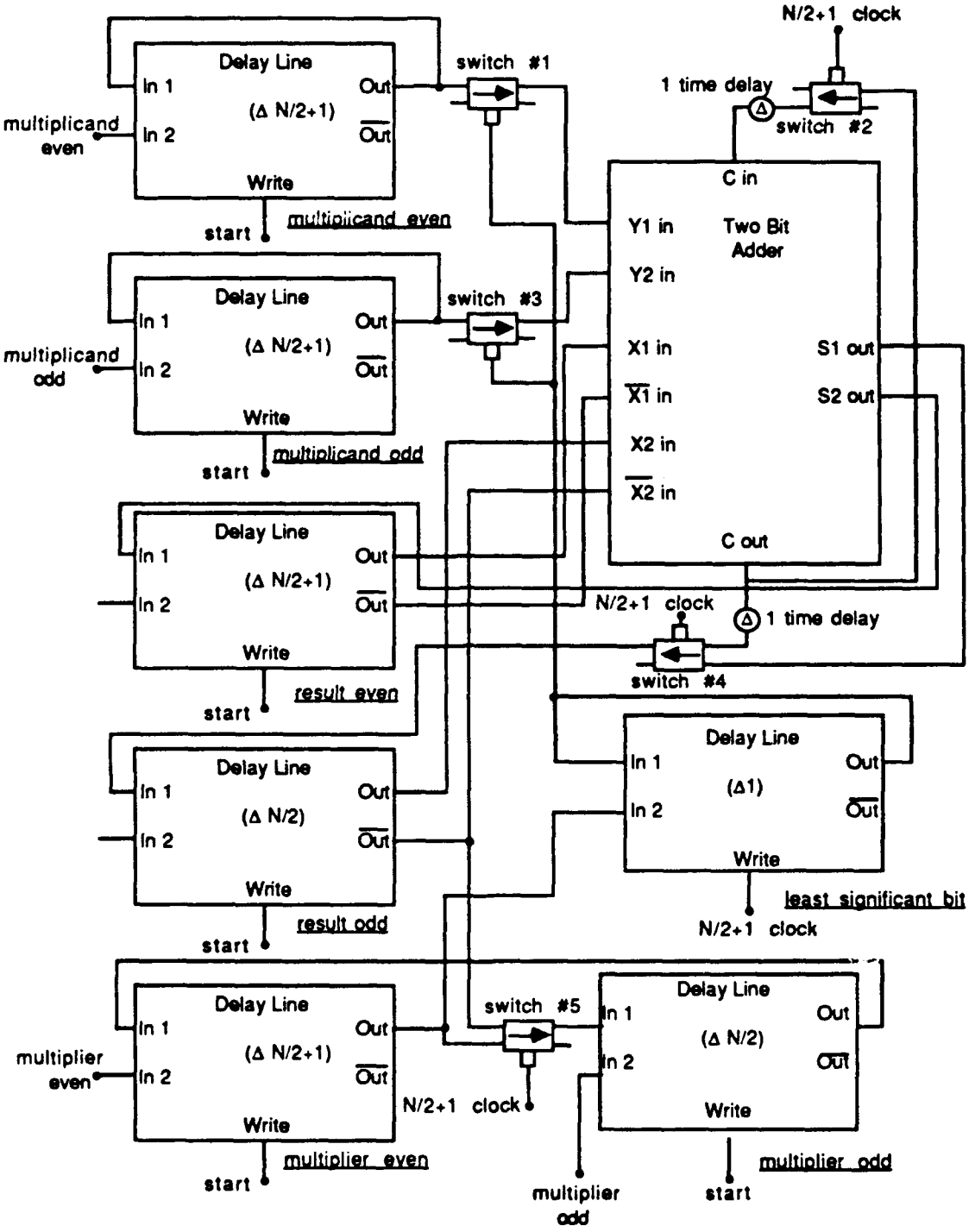


Figure 9. Two Bit Odd/Even Multiplier.

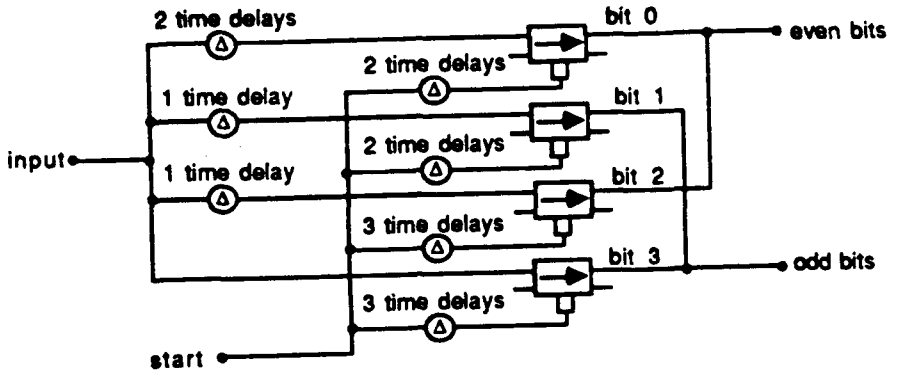


Figure 10. Example Circuit to Split 4 Bit Number into Odd and Even Bits.

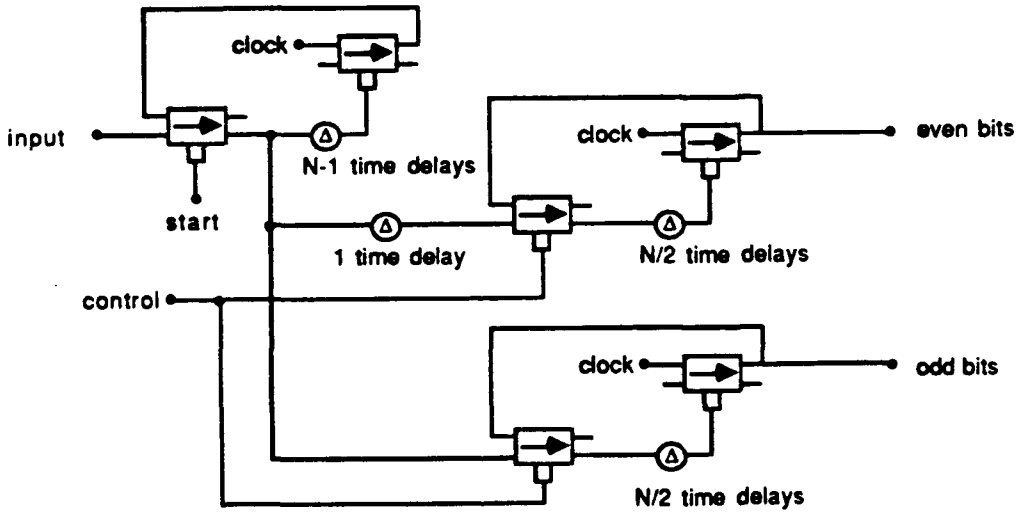


Figure 11. Circuit that Splits Numbers in Odd and Even Bits Using a Delay Loop.

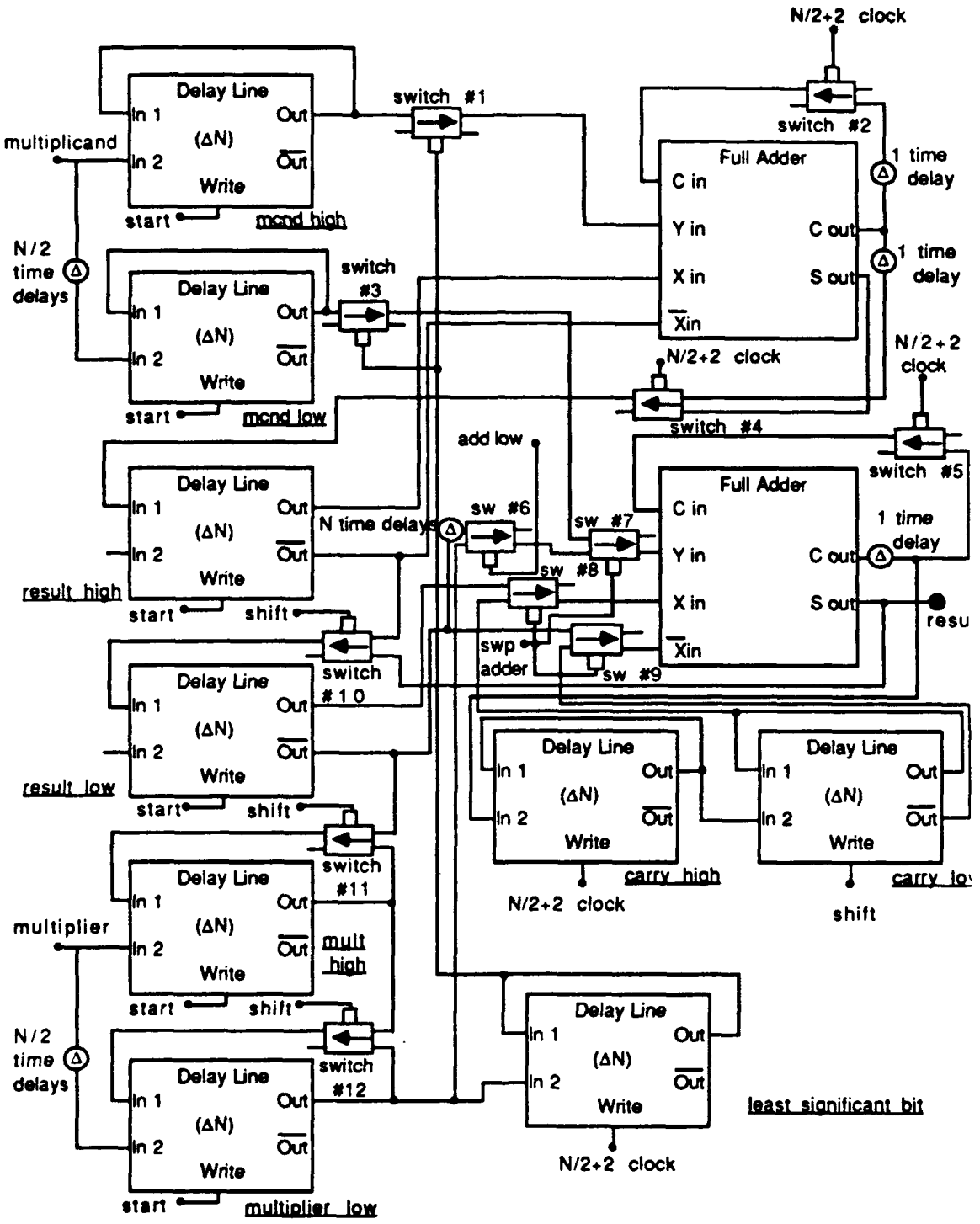


Figure 12. Two bit high/low multiplier.

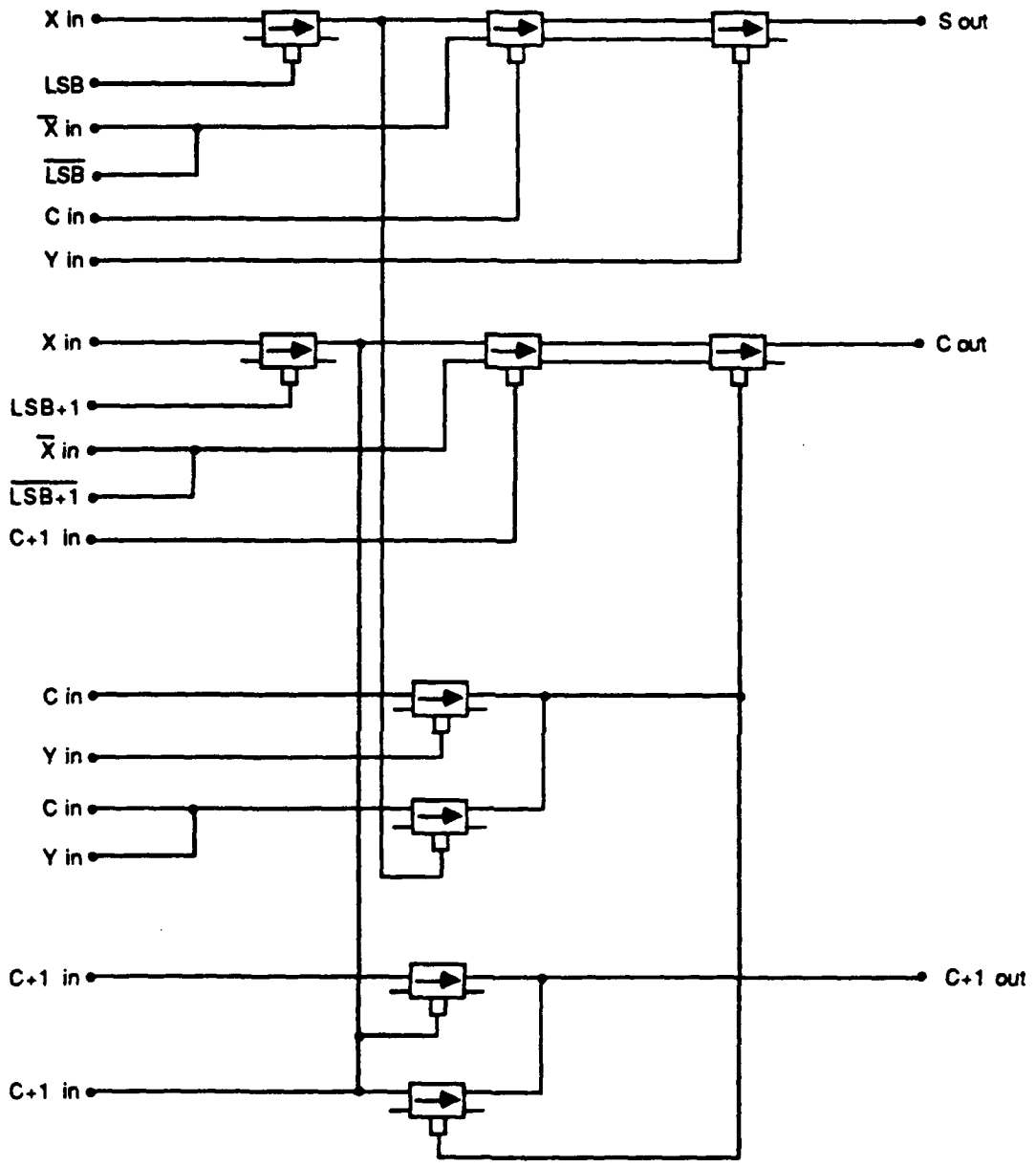


Figure 13. Adder which can add a multiple of the first addend to the second addend.

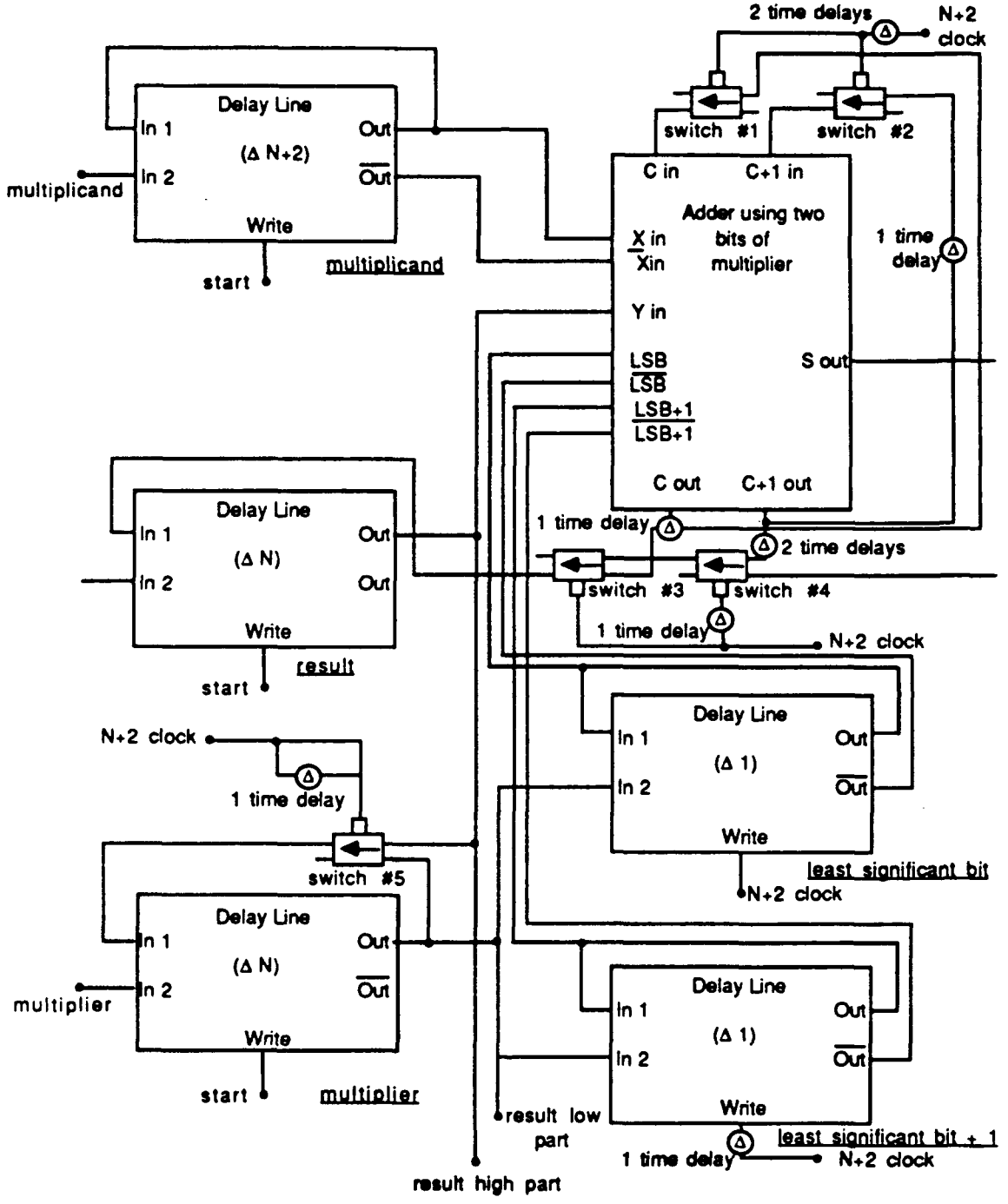


Figure 14. Multiplier using two bits of multiplier at a time.

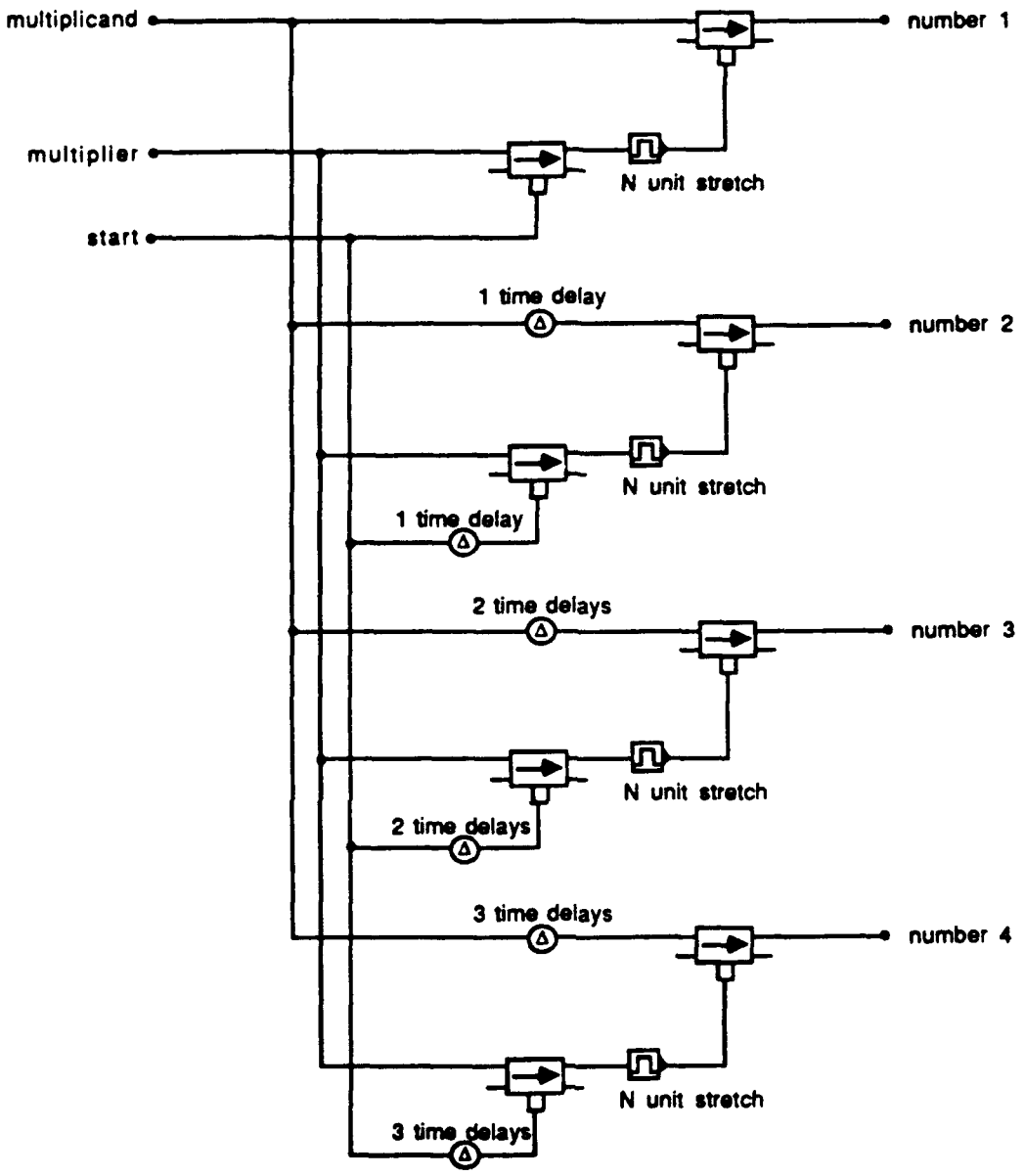


Figure 15. Generation of all shifted versions of the multiplicand and bits required for counting columns.

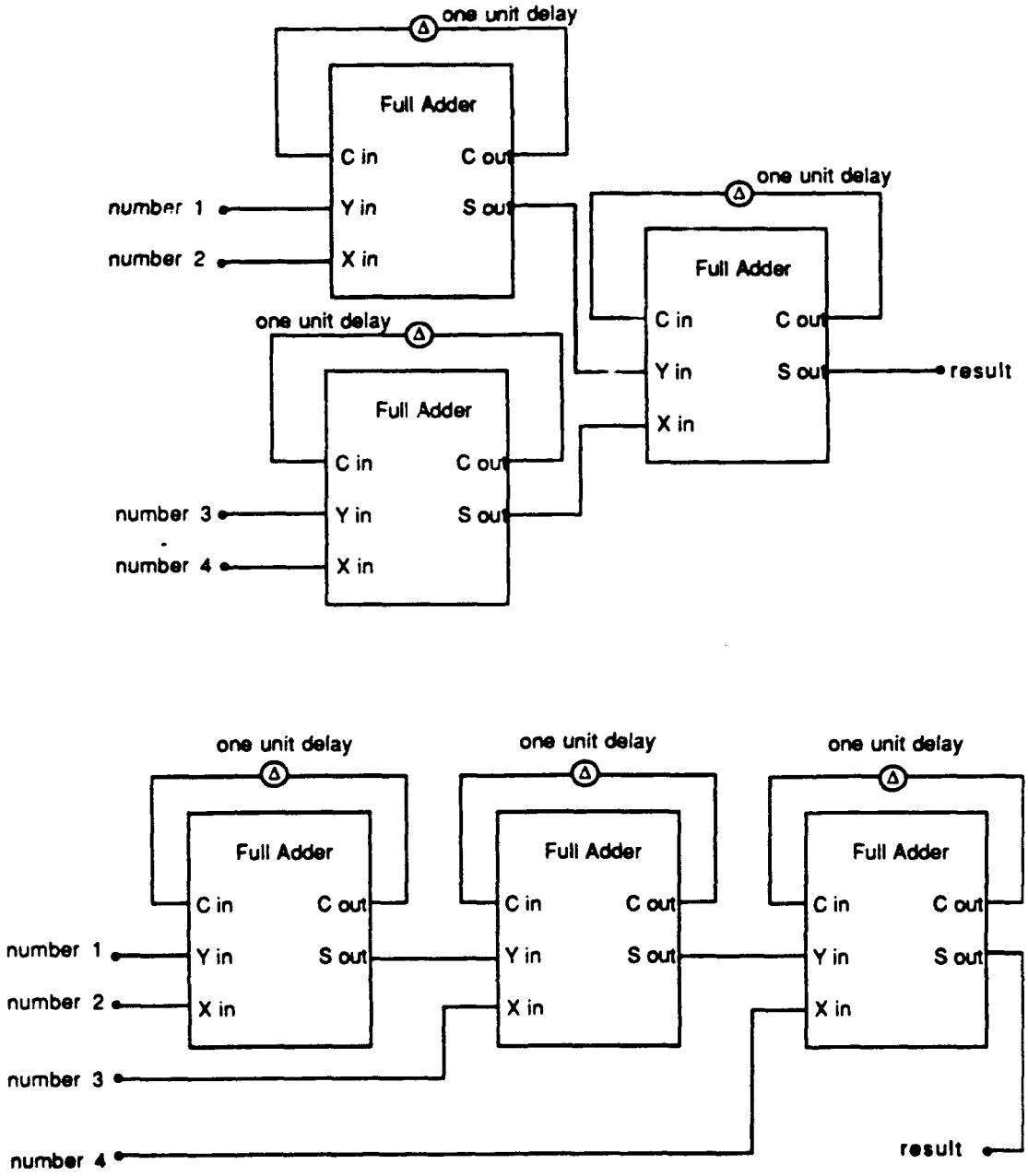


Figure 16. Two Adder Connections for the Simple Pipelined Multiply.

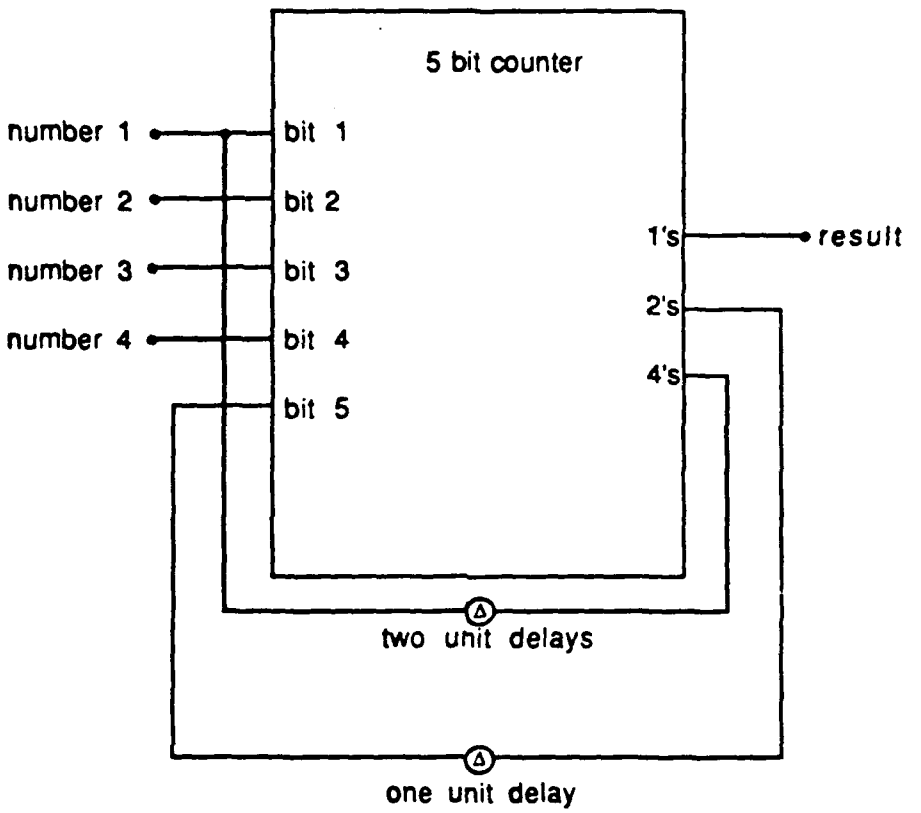


Figure 17. Implementation of Column Counter Multiply.

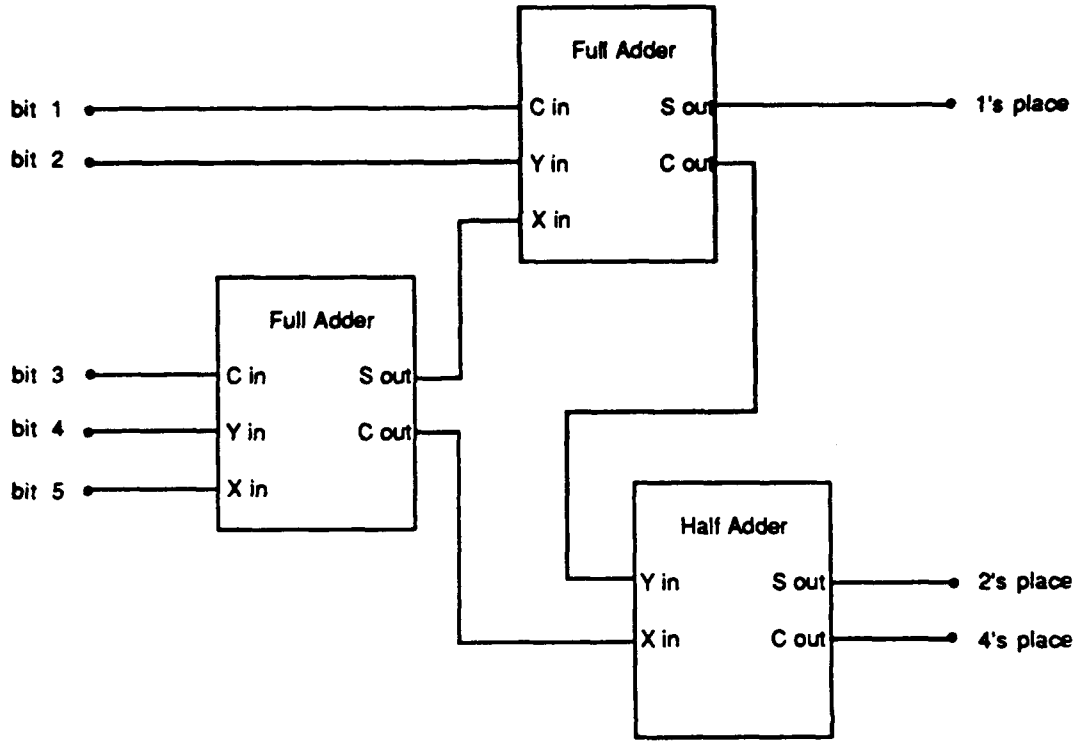


Figure 18. Implementation of a five input counter using half and full adders.

Table 1. Comparison of Multipliers.

Multiplier	Number of Switches	Time for Multiply
Single Adder	21	$(N+1)^2$
Two bit odd/even with no conversion	32	$(\frac{N}{2}+1)(N+1)$
Two bit odd/even conversion with varying delays	$32+2N$	$N(\frac{N}{2}+4)$
Two bit odd/even conversion with looping	44	$N^2+N(\frac{N}{2}+1)$
Two bit high/low carries added on next iteration	51	$N(\frac{N}{2}+2)$
Two bit high/low carries added at end	44	$N(\frac{N}{2}+2)$
Multiplier taken two bits at a time	30	$(N+2)(\frac{N}{2}+1)$
Simple Pipelined Approach	$6.5N$	On-Line
Counting Columns	$= 2N+4(N+\log_2 N)$	On-Line

Table 2. Multiplier Results for 16 and 32 bit numbers.

Multiplier	16 Bit Word		32 Bit Word	
	Number of Switches	Time	Numbe of Switches	Time
Single Adder	21	289	21	1089
Two bit odd/even with no conversion	32	153	32	561
Two bit odd/even conversion with varying delays	64	204	96	640
Two bit odd/even conversion with looping	44	400	44	1568
Two bit high/low carries added on next iteration	51	160	51	576
Two bit high/low carries added at end	44	160	44	576
Multiplier taken two bits at a time	30	162	30	578
Simple Pipelined Approach	105(97)	On-Line	208(193)	On-Line
Counting Columns	112	On-Line	221	On-Line

Appendix A

Hatch Version of a Single-Bit Multiplier

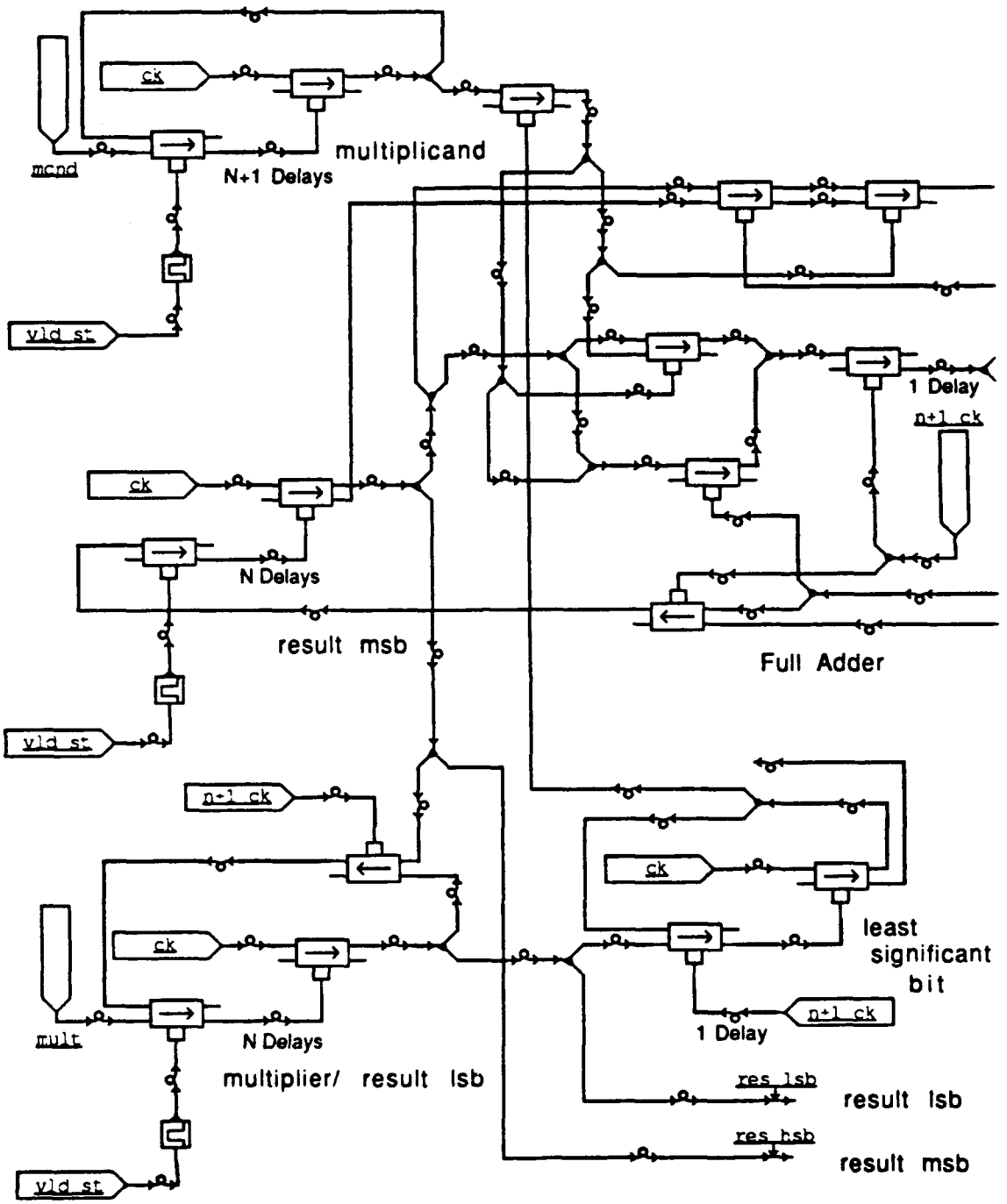


Figure A-1. Hatch Version of a Single-Bit Multiplier.

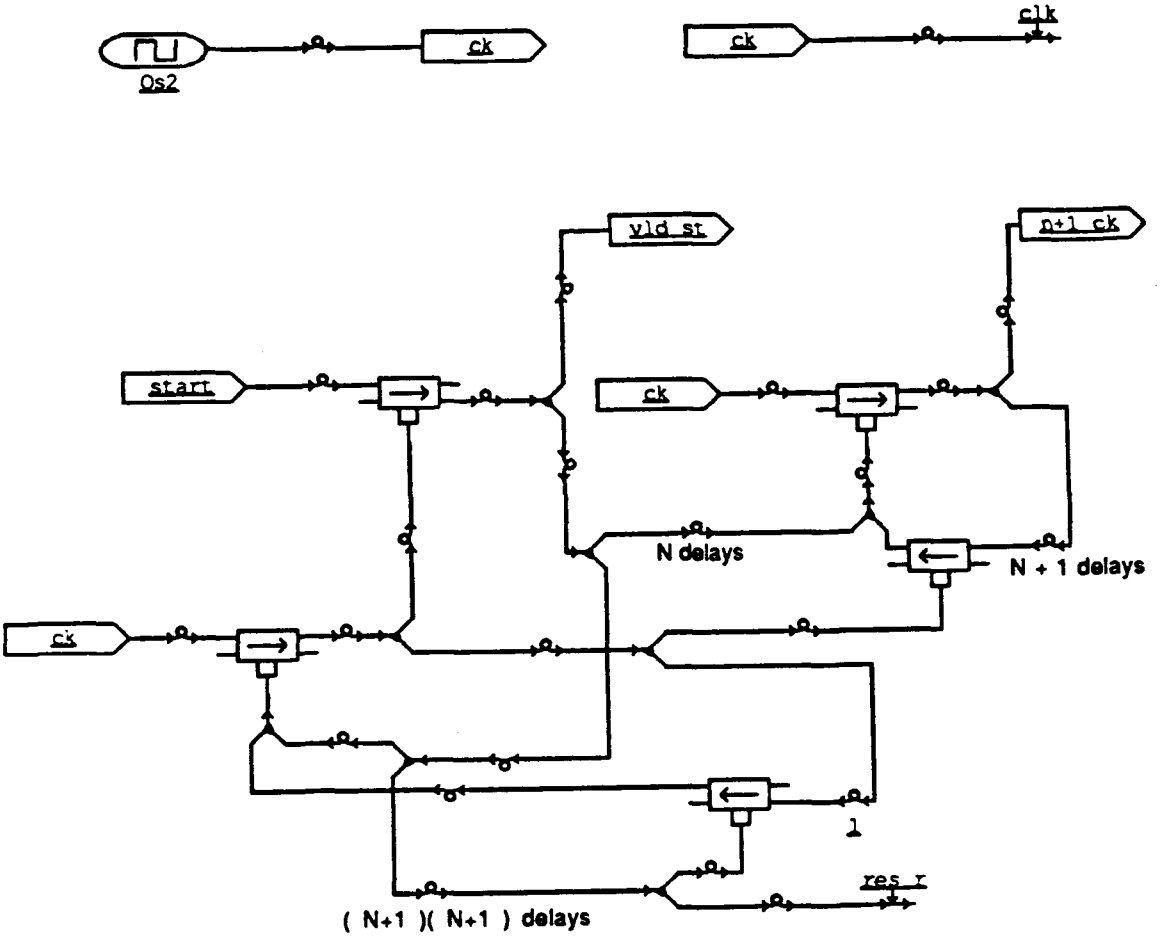


Figure A-2. Timing and Control Circuitry.

Appendix B

Hatch Version of a High/Low Two-Bit Multiplier

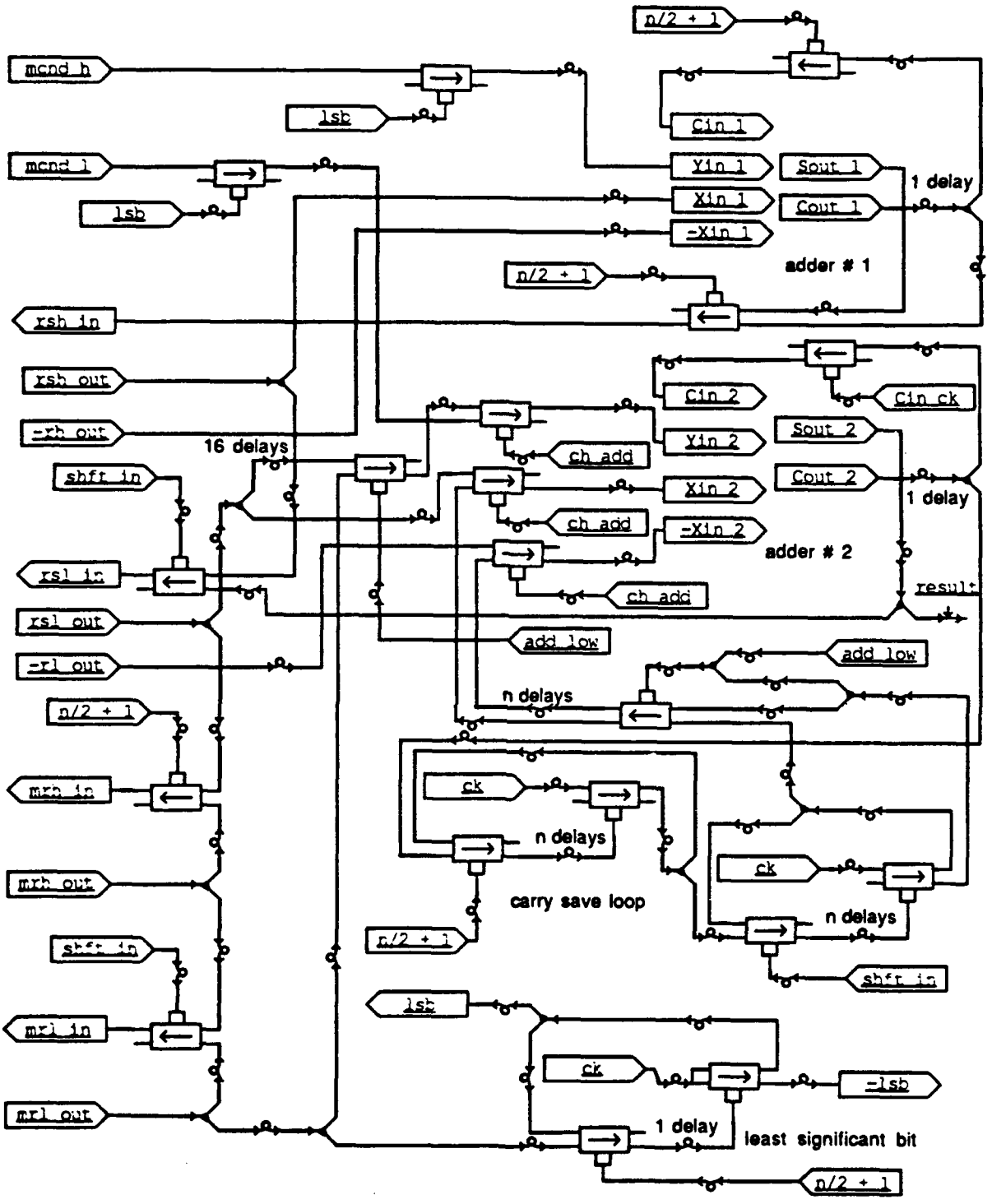


Figure B-1. Hatch Version of a High/Low Two-Bit Multiplier.

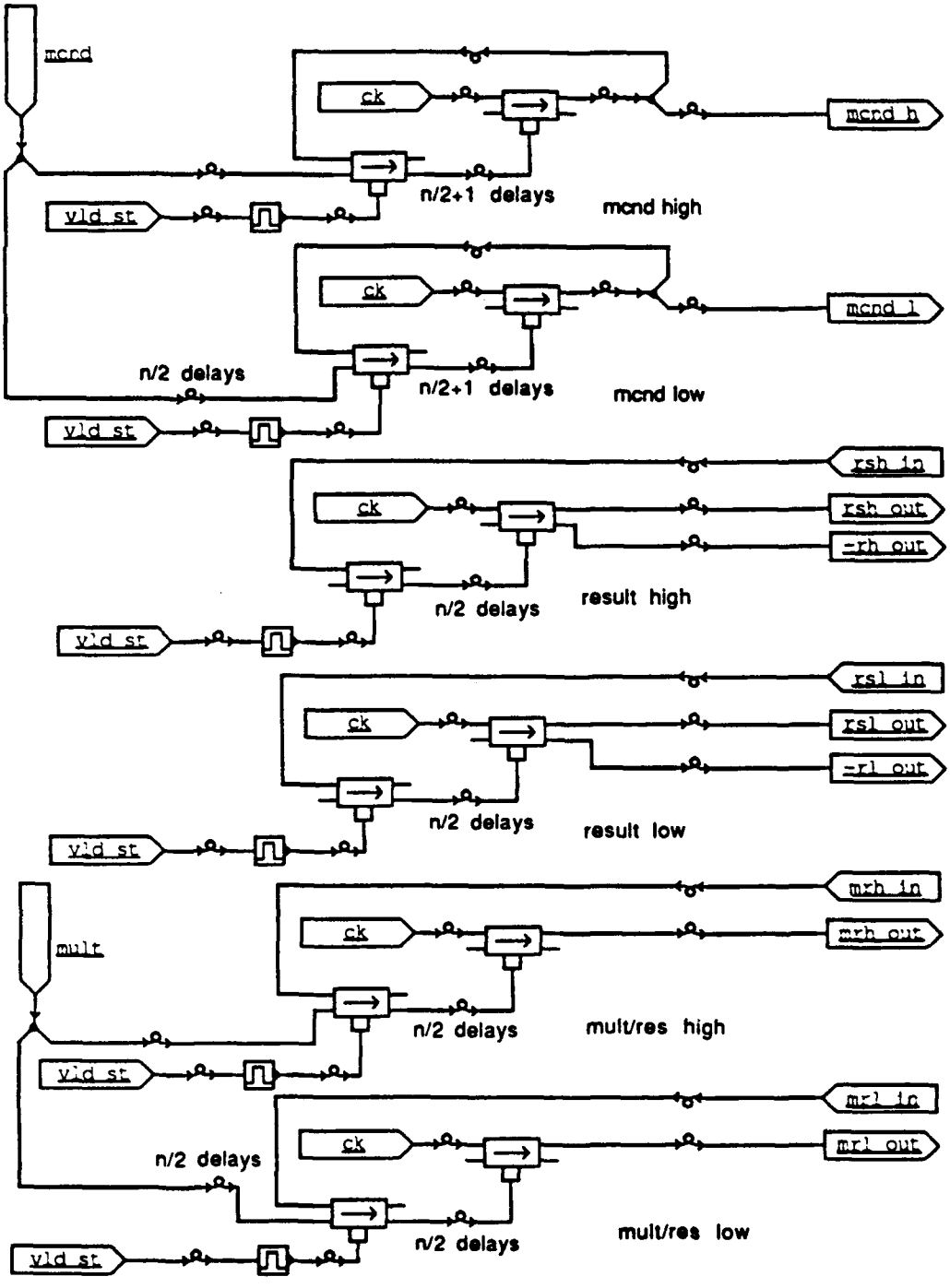


Figure B-2. Delay Lines for the High/Low Multiplier.

Appendix C

Hatch Version of a Multiplier Using Two-Bits of the Multiplier at a Time

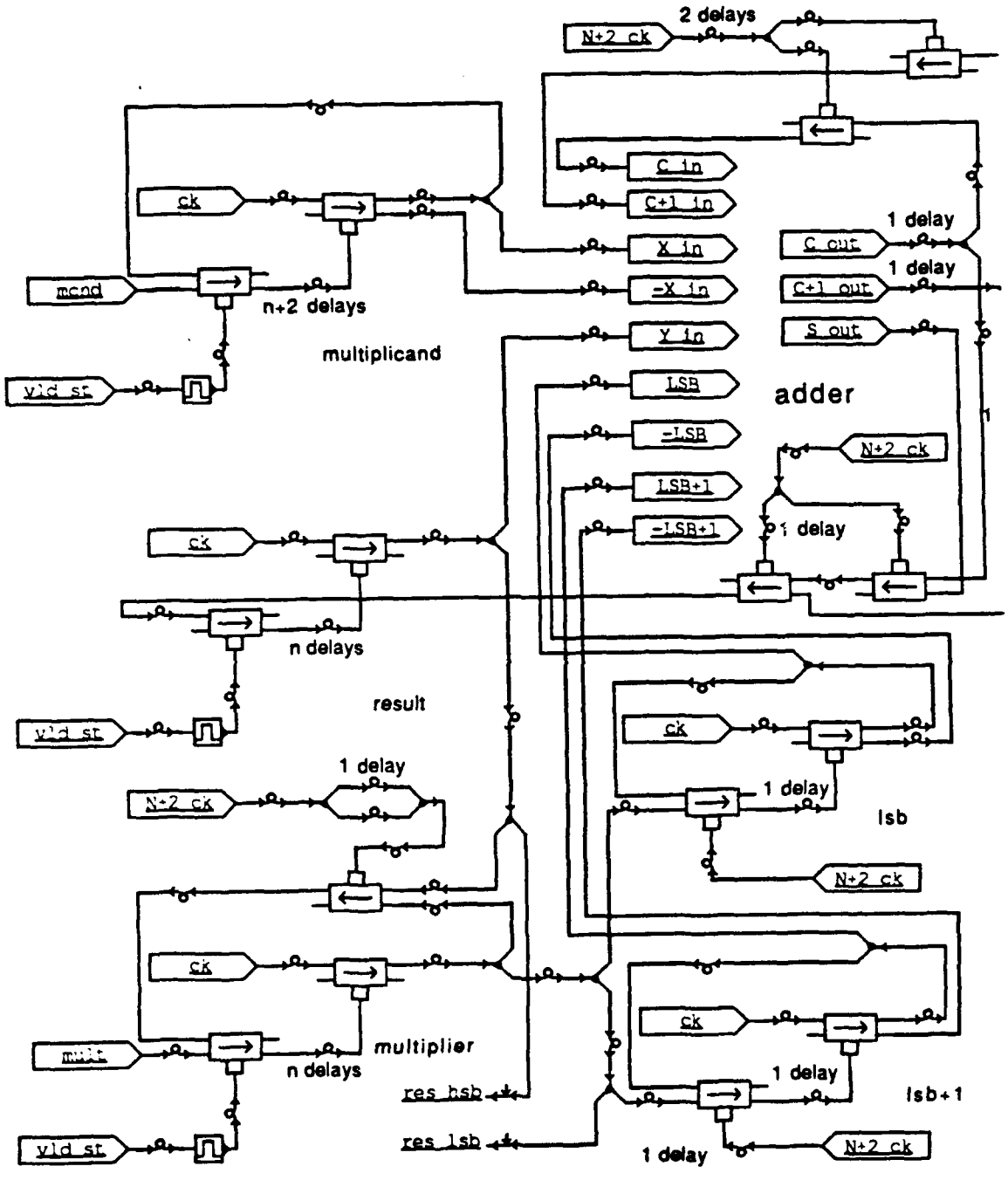


Figure C-1. Hatch Version of a Multiplier Using Two-Bits of the Multiplier at a Time.

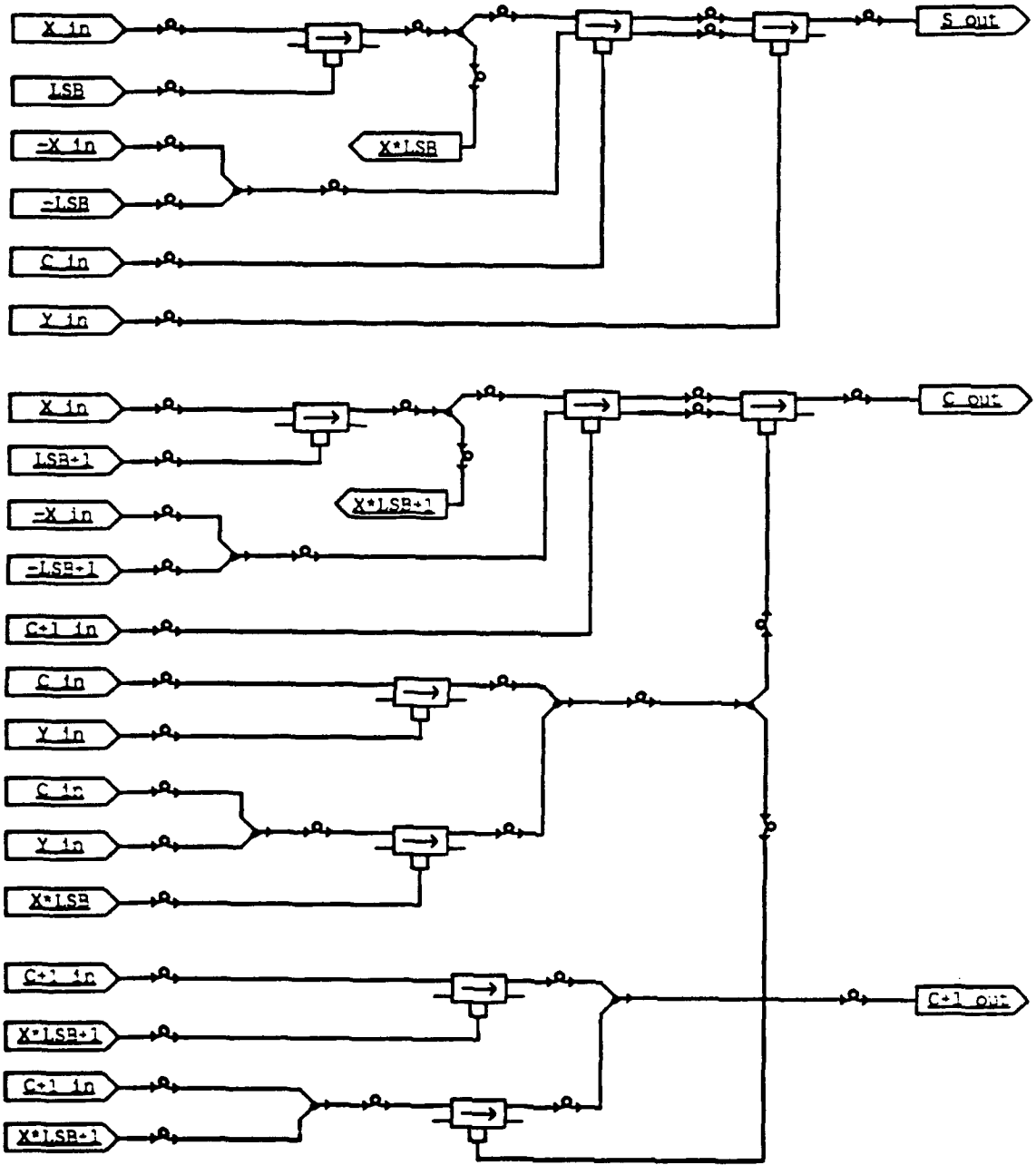


Figure C-2. Adder Circuit for Multiplying by Two-Bits of the Multiplier.

ATTACHMENT B

A Bit-Serial Optical Divider

by

Michael P. Dickman
Andrew R. Pleszkun

Optoelectronic Computing Systems Center
Univserity of Colorado
Campus Box 525
Boulder, Colorado 80309-0525

August 1990

The Center is sponsored in part by the National Science Foundation Engineering Research Grant number ECD 9015128 and by the Colorado Advanced Technology Institute (CATI), an agency of the State of Colorado. CATI promotes advanced technology education and research at universities in Colorado for the purpose of economic development. The work detailed in this reported was primarily supported by CALSPAN under Task No. P-9-6009 Delivery No. 0009 for the Rome Air Development Center (RADC).

A Bit-Serial Optical Divider

Michael Dickman and Andrew R. Pleszkun

Center of Optoelectronic Computing Systems
Department of Electrical and Computer Engineering
University of Colorado
Boulder CO 80309-0525

Abstract

This paper considers the design of a bit-serial divider using optical devices. The first stage of this project, the design of a multiplication unit, was described in a previous discourse [PlDi89]. This paper concentrates on the development of the next portion of the functional unit, i.e. division. The optical devices used for the implementation of the divider are assumed to be the same as those specified for the multiplier. All designs, therefore, will use optical fibers, splitters, connectors, pulse stretchers, and five port 2x2 switches. Furthermore, the inputs and outputs, like those to the multiplier, are represented bit serially, least significant bit first.

This paper describes several different division methods and discusses their applicability to simple bit serial implementation. Several different designs are then presented and compared. Square rooting, an operation very similar to division is then examined. Finally, a combined divider/square rooter, a divider/multiplier/square rooter and a divider/multiplier are described.

1. Comments on Division

Division differs from multiplication in two major ways. For one, the bits of the result are produced most significant bit first. Since we are assuming that the input and output bits are to be presented least significant bit first, as in the SCAMP bit serial computer [HEJP88], the implementation of an on-line divider is impossible. One can, however, design a most significant bit first on-line divider [TrEr77]. Secondly, each iteration of the division process is dependent upon the previous one. That is, each iteration must be completely finished in order to determine the actions of the next iteration. This makes division extremely difficult to parallelize. In general, division presents a much more difficult problem to solve than multiplication. The following sections present the designs of several types of dividers.

2. Shift and Compare Division

When one divides two numbers by hand using long division, shift and compare division [Wake81] is being performed. Consider the following example:

```
        101
    ------
110 ) 11110
      -110
      ----
       110
      -000
      ----
       110
      -110
      ----
         0
```

The first step in accomplishing this division is to determine the most significant quotient bit (which in this case is a one). This bit is determined by comparing the first three digits of the dividend (11110) with the three bits of the divisor (110). Since the three most significant bits of the dividend are larger than the divisor, the first quotient bit is set to one and the divisor is subtracted from the dividend. One then compares the divisor with the next three bits of the dividend. In our example, the divisor is larger, the next quotient bit is determined to be zero, and nothing is subtracted from the dividend. This process is repeated again and the final quotient bit is determined

to be a one. One can then perform the final subtraction. The result of this operation, which is 0, provides the remainder of the division. This example, which happens to be in binary, is accomplished in the same way that a decimal division would be performed. This technique, like all other division methods, produces the quotient bits most significant bit first.

The relative simplicity of this division technique makes it a likely target for implementation using optical devices. Since the division consists only of several simple iterations or steps, relatively little control is needed. The other hardware required includes a subtractor, a comparator, and storage for three data items. The subtractor and comparator can actually be replaced by a single adder and some logic to determine the sign of a two's complement addition. Thus, the hardware required to implement the shift and compare division is minimal. In fact, as will be shown later in this paper, such a divider can be implemented with as few as seventeen 2x2 switches.

3. SRT Division

SRT division [Atki67,Tayl85] is similar to the shift and compare division, with the exception that instead of determining a single quotient bit every iteration, one attempts to determine several bits. Thus, in the previous example, one might compare the three bits of the divisor with the top four bits of the dividend. Then, instead of determining if the divisor is simply larger, one must determine how many times the divisor will fit into the dividend. In the case of our example, the divisor fits into the dividend two times, the top two quotient bits would then be set to 10, and two times the divisor would be subtracted from the dividend. One might also determine the entire quotient in a single iteration by determining that the divisor fits into the complete dividend five times.

By producing several quotient bits at a time, one can reduce the number of compare and subtract iterations required, and thus, improve the overall speed for division. However, there is a severe penalty in complexity. In order to determine the number of times that one can fit the

divisor into the dividend one needs to use a look-up table or actually compare the dividend with multiples of the divisor. Such a table would require N×N or N^2 entries (where N is the number of bits in the dividend) to calculate the correct quotient bits. One can use an approximate look up table provided that any errors made in the guess of the quotient bits can be made up using some type of redundant code. Once the quotient bits are determined, a multiply is required to determine the exact value to be subtracted from the dividend. Once this subtraction is completed, we iterate in the same way until the complete quotient is determined. It may then be necessary to convert any redundant coded values into standard binary.

The above description is intended only to give the reader an idea as to the complexity of using SRT division techniques. The implementation of such a divider would certainly require a great deal of hardware, and thus, would not be reasonable to implement with the present technology of optical switches. SRT division, therefore, will not be considered in the implementation of the arithmetic functional unit.

4. Reciprocal Approximation

Division using reciprocal approximation [AEGP67] is carried out by determining the reciprocal of the divisor and multiplying it with the dividend. This is described as follows:

$$\text{Dividend/Divisor} = \text{Quotient}$$

$$\text{Reciprocal} = 1 / \text{Divisor}$$

$$\text{Quotient} = \text{Dividend} \times \text{Reciprocal}$$

The difficulty of this technique lies in determining the reciprocal of the divisor. This reciprocal can be determined to be the product of several terms $R_1, R_2, R_3, \cdots R_n$ such that:

$$\text{Reciprocal} = R_1 \times R_2 \times R_3 \times \cdots \times R_n$$

$$\text{Divisor} \times R_1 \times R_2 \times R_3 \times \cdots \times R_n = 1$$

$$\text{Quotient} = \text{Dividend} \times R_1 \times R_2 \times R_3 \times \cdots \times R_n$$

Each one of these product terms can be determined with a multiplication and $\log_2(N)$ terms are required to produce a close approximation to the reciprocal, where N is the number of digits in the divisor. One must also multiply these product terms with the dividend to determine the quotient. By using this division technique, one can reduce a single division into $2\log_2(N)$ multiplications. Practical reciprocal approximation dividers utilize a table look-up to reduce the number of multiplications required to perform a division. These devices use the divider and dividend to look up an approximate reciprocal in a table. Thus, the calculation of the reciprocal can be reduced into a look-up and only a few, reciprocal correcting, multiplications. Unfortunately, such a divider still requires some multiplication and an expensive table look-up.

Reciprocal approximation is a valuable method of division only when very fast parallel multipliers or cheap table look-up operations exist. Since this is certainly not the case for bit serial optical devices, reciprocal approximation does not appear to be a valid choice for use in the implementation of any reasonable bit serial divider. Like SRT division, reciprocal approximation will not be considered further.

5. Restoring Division Using Addition

The basic shift and compare algorithm involves a compare and then a conditional subtraction. Comparisons are accomplished, in most systems, by actually performing a subtraction and determining the sign of the result. Most division algorithms perform the subtraction and the comparison at the same time. If the result of the subtraction is negative, the subtraction should never have been carried out, and some type of correction is required. The restoring divider corrects this error by adding the divisor (the value that was erroneously subtracted) back into the dividend, thus restoring it to its correct, original value.

The restoring divider, like all dividers in this paper, divides a 2N bit dividend by an N bit divisor producing an N bit quotient and an N bit remainder, where N is the word length. Each iteration involves a subtraction of the N bit divisor from the top N+1 bits of the dividend. The

top bit of the dividend is only used to determine the sign of the result, thus, only an N bit subtraction is carried out. The first step of a single iteration is to shift the dividend left, saving its most significant bit to determine the sign of the subtraction. The difference between the remaining top N-bits of the dividend and the divisor is then determined. If the result is negative, the current quotient bit (the most significant bit on the first iteration) is set to zero, and the divisor is added back into the dividend. If the result of the subtraction is positive, the quotient bit is set to a one, and no correction is made. The dividend is then shifted again and the iteration repeated. After N iteration, the final quotient is complete and the result of the final subtraction (and possible restoration) yields the appropriate remainder.

A simplified block diagram of the restoring divider is presented in Figure 1. This figure, like the others included in this paper, contains a 2N clock. This is simply a device which produces a pulse every 2N cycles. The remainder of the figure shows the divider. The top delay line holds the divisor and the lower delay line holds the dividend. The dividend delay line also holds the bits of the quotient which replace the exhausted most significant bits of the dividend. The dividend delay, although not entirely obvious, is 2N+1 bits long. The extra bit length facilitates the shifting of the dividend after every iteration. The divisor delay line is N bits long. This means that every N cycles, the divisor is again presented to the adder (which is twice per iteration since each iteration is 2N bits long). During the first N cycles of the iteration , the divisor is subtracted from the dividend. After this time, the sign of the result is determined and the restore signal is set accordingly. If restore is set high, the divisor is added back into the dividend during the next N cycles and switch #1 selects this restored dividend to send back into the storage delay line. If the restoration is not required, the original subtracted dividend is returned to its delay line. Since only the top half of the dividend is restored, the lower half of the dividend must pass unaltered back to the dividend delay line. This is accomplished through the N bit delay loop that connects to switch #2. Switch #3 is used to switch the current quotient digit into the correct location of the dividend delay line, thus replacing the old most significant bit of the dividend.

The implementation of this divider requires the use of only fifteen switches, two pulse stretches, and a 2N timing clock. Pulse stretchers, may in fact, require two switches each for their implementation. Furthermore, the 2N clock may require up to five switches for its construction. This brings the total switch count up to twenty four switches. This divider completes a problem after N iterations, each of which is 2N cycles long. This means that this divider circuit, ignoring any possible data set up time, can complete a division in 2N×N cycles.

6. Restoring Division Using Copies

The previous divider restores the dividend by adding the erroneously subtracted divider back into it. Another method of restoring the dividend is to actually keep another copy of the dividend which is not altered by the subtraction. Then, instead of conditionally adding the divisor back into the dividend, one simply selects the correct copy of the dividend. Since, storing a copy of a data item in a delay line is relatively cheap, this turns out to be a good way to implement restoring division.

In the previous example, each iteration was required to perform, in the worst case, a subtraction as well as an addition every iteration. Since each of these operations is N-bits long, a single iteration is required to be 2N cycles long. The copy method, on the other hand, only requires a single N-bit subtraction. By splitting the dividend into a high part and a low part, one can access the dividend every N cycles. This allows a subtraction, restoration iteration to be completed in only N cycles.

Figure 2. illustrates a possible design for a copy restoring divider. The divisor, high dividend and low dividend are all stored in delay lines. The dividend delay lines are one unit longer to cause the left shift every iteration. The quotient, as in the previous example, is shifted into the dividend delay line replacing the used bits. Each iteration consists of the subtraction of the divisor from the top bits of the dividend. One bit must actually come from the low dividend delay line, and appended as the least significant bit by switch #1. The sign of the difference is

then calculated and stored into the current quotient digit delay line. This bit is used to determine if the subtracted dividend or the original 'restored' dividend is to be selected for the next iteration. This selection is carried out by switch #2. It might be noted that the subtracted version of the dividend is stored in the original high dividend delay loop and the original version is stored in the loop directly below this high loop.

As mentioned earlier, this divider requires N iteration of N cycles each. Thus, a complete division is carried out in only N×N cycles. This copy implementation of division requires nineteen switches plus the timing clock for a total of twenty four switches. Since this restoring version is much simpler, quicker, and quite possibly cheaper, it is certainly recommended over the previous division implementation.

7. Non-Restoring Division

Restoring division is based on the idea that the dividend should always be restored to its correct value prior to the subtraction of the divisor. It is possible, however, to eliminate this restoration provided that one knows the sign of the result, and thus, the correct quotient digit. If the sign of the result is positive, one simply subtracts the divisor from the correct dividend. Thus, this case is treated the same as for a restoring division. If, on the other hand, the difference is negative, a different approach is taken. The dividend is left in its unchanged negative state. On the next iteration, the divisor is added into the dividend, instead of being subtracted. The difference is then evaluated on the same basis of being positive or negative, and the cycle is continued. After N cycles, the correct quotient is determined. The remainder, however, may not be correct. If the last quotient digit is a zero, then the divisor must be added back into the dividend, thus producing a valid quotient.

Figure 3. displays a possible implementation of a non-restoring divider. Like the copy restoring division, the dividend is broken into its high and low parts in separate delay lines. The divisor and current quotient bit are also contained in separate delay lines. The complete quotient,

however, gets shifted into the dividend delay loop replacing the used most significant bits. Switches #1 and #2 are used to transform the adder into either an adder or a subtractor depending upon the sign of the last result which is represented by the last quotient bit. Switches #3 , #4 and #5 are used to multiplex the adder so that the final remainder can be adjusted if the quotient digit is zero. Switch #6 ensures that no adjustment will occur if the final quotient digit is a one.

This non-restoring divider, like the second restoring divider, calculates a division in N×N cycles. However, it requires five extra switches and a pulse stretcher, thus resulting in a total of thirty switches. A non-restoring divider, therefore, is not a practical means of division using optical devices.

8. Parallelizing Division

Up until this point we have considered strictly bit serial implementations, manipulating only a single bit at a time. This section considers the possibilities and problems of attaining some parallelism while performing a division operation. The same ideas as were examined for multipliers will be briefly explored. Unfortunately, due to the nature of division, most of these strategies are inefficient or inapplicable.

The first method to be examined is splitting the bits up between its odd and even bits. In this manner, one can operate on two sequential bits in parallel at the same time. Thus, instead of using a single bit adder, a two bit adder is required. This method does, in fact, work. However, as was noted for multiplication, splitting the bits up between their odd and even components is either far to expensive or much to time consuming to be worthwhile. This method would, however, be effective if the bits were to be broken up into N separate latches, thus, fully parallelizing the additions and subtractions. Such a device, being impractical given the current technology, could compute a division in N cycles and compute a quotient nearly on line most significant bit first.

The second method to be considered is splitting the bits between their high and low parts. In this manner, the expensive splitting of the bits is avoided. This method worked well for multiplication during which the carry out of the low order bits could be added in during the next iteration. In division, however, all the carries are required at the end of each iteration in order to determine the next action. This renders the high/low division of bits impossible.

The final method, the one most effective for multiplication, is recoding of the quotient. This turns out to be SRT division during which several bits of the quotient are selected every cycle. As was pointed out earlier, such a scheme is simply not practical.

As it turns out, searching for possible parallelisms while evaluating division is not very practical for our purposes. Division can be parallelized to the extent of performing each iteration in one parallel step. Each iteration, however, is dependent upon the previous iteration removing the possibility of further parallelization. This turns out to be the major difficulty in implementing very high speed dividers in any technology.

9. Square Rooting

Performing a square root and performing a division require the accomplishment of essentially the same tasks [Maje83]. For this reason, it is not difficult to build a device that does both division and square rooting. The following discussion explains a square root algorithm and presents a possible implementation.

A square rooting algorithm can be described by the following recurrences:

$$X[i] = X[i-1] - (2Y[i-1] + y[i]2^{(n-i)})y[i]2^{(n-i)}$$

$$Y[i] = Y[i-1] + y[i]2^{(n-i)}$$

$$y[i] = 1 \text{ or } 0.$$

For these recurrences i runs from 1 to n, X[0] is the number to be square rooted, and Y[i] accumulates the square root. The y[i]'s contain the value of the current quotient bit such that y[1] is the most significant bit and y[n] is the least significant bit. We first guess that the value of the

most significant bit of the square root, y[1], is a one. This leads to the initial square root guess of 100..0. X[i] is then calculated by subtracting the square of this guessed value from the original X value. If X[i] turns out to be negative, y[i] is changed to a zero and X[i] is recalculated. In fact, X[i] is simply restored if the result is negative because X[i] = X[i-1] when y[i] = 0. We then update the square root, Y[i], by adding the current quotient digit, y[i], into the appropriate location of Y[i]. The next iteration then involves guessing the next bit of the square root. X[i] is again determined by subtracting the additional amount caused from squaring this new square root guess from the previous X[i]. Like division, square rooting requires a comparison, a subtraction, and a shift. The primary difference between the two functions is the generation of what value must be subtracted from the number or dividend. This value is simply the constant dividend in the case of division. In square rooting, the number to be subtracted varies with the final quotient digits selected.

The square root equivalent of a divisor can be generated mechanically given the current quotient digits. That is, the seemingly complex expression, $(2Y[i-1] + y[i]2^{(n-i)})y[i]2^{(n-i)}$, which must be subtacted from X[i−1], can be easily generated, given Y[i−1] and the fact that y[i] = 1. The following example which calculates the square root of 49 illustrates this fact:

$$
\begin{array}{r}
111 \\
\sqrt{110001} \\
-10000 \\
\hline
100001 \\
-10100 \\
\hline
1101 \\
-1101 \\
\hline
0
\end{array}
$$

Every iteration the current quotient with a one generated in it is subtracted from the number. Furthermore, the quotient digits are shifted right one place with respect to the number, and the generated one is shifted right two places with respect to the number that is being square rooted. During the first iteration, the current quotient is zero and a one is generated and sub-

tracted from the second most significant bit. During the next cycle, the one bit of the quotient is subtracted from the second most significant bit, and the generated bit is shifted right two places and subtracted from the fourth most significant bit. The final iteration involves subtracting the two ones of the quotient from the third and fourth most significant bits (shifted right once), and the generated bit from the sixth most significant bit.

An implementation of a bit-serial square rooter is shown in Figure 4. The number to be square rooted is stored in the top delay line, its copy for restoration is stored directly below that, and the loop which holds the quotient digits is located at the bottom of the diagram. The current quotient bit also has its own loop so that it can select the correct version of the number being square rooted. Switch #1 is used to generate the extra bit to be subtracted. Switch #2 is used to place the current quotient bit into the quotient delay loop at the appropriate time.

This square rooter can perform its function in N iterations of 2N cycles, or 2N×N total cycles. Furthermore, its design is fairly simple, requiring only fifteen switches. However, Two separate clocks are required for control. One clock is required to set the carry for each iteration and to shift the numbers into the quotient loop, and one clock is required to generate the extra subtraction bits. Both of these clocks can easily be implemented using seven switches, bringing the total number of switches up to twenty-two.

10. A Divider/Square Rooter

The square rooter mentioned in the previous section can easily be extended to enable it to perform the division operation. Since the square rooter performs all the tasks necessary for division, only simple modifications of the circuit are required. The block diagram of the divider/square rooter is presented in Figure 5. One extra delay line is added to store the quotient. This is simply because the quotient digits can not be stored with the digits of the divisor. The divisor is stored where the square root quotient is stored when a square root operation is being performed. One other switch is required to turn off the 'generate a one signal' when a division is

taking place. The rest of the circuit behaves exactly as it does for the square rooter. It should be noted that the divisor is subtracted from all 2N bits of the dividend every cycle. Since the divisor is only N bits long, the rest of the loop must be padded with zeros. This padding of the loop may, in fact, cost an extra switch depending upon the input interface.

11. A Multiplication, Division, and Square Root Unit

Adding multiplication to the abilities of the divider/square rooter can be accomplished with as few as six extra switches. Most of these switches are required to multiplex the quotient delay line and to convert the subtractor to a full adder. Additional switches would be required if a unique input or output terminal is needed for all three operations. In the last last paper, several multipliers were examined. The sequential devices that were considered looked at the multiplier least significant bit first. If this bit was a one, the multiplicand was added into the least significant portion of the partial product. The partial product was then shifted left and the process repeated. The divider square rooter lends itself to the implementation of a similar multiplier with one minor change. Instead of looking at the least significant bit first, it is more appropriate to examine the multiplier most significant bit first. By using this approach, one can add the multiplicand into the most significant portion of the partial product and then shift the multiplicand left.

The complete multiplier, divider, and square rooter is presented in Figure 6. The top delay line holds the result, the middle loop holds the multiplicand, and the bottom one contains the multiplier. The first two delay lines are the appropriate length. Unfortunately, since the multiplier must be accessed MSB first, an extra fiber length in this loop is required. Switch #1 adds this extra fiber into the loop during a multiplication, thus ensuring that the multiplier shifts left every iteration. Switches #2 and #3 are required to multiplex this multiplier delay line so that it can be used in both multiplication and division. It would cost the same amount (in switch count) to simply create a new delay loop used only for multiplication. Switches #4 and #5 are used to convert the subtractor to a full adder. Finally, switch #6 is used to ensure that nothing is added to

the partial product when the current multiplier bit is a zero. It should be noted that the current quotient bit delay line for division and the square root operation simply holds the current most significant bit for multiplication.

12. A Divider/Multiplier

In many situations the square root operation may be an unnecessary expense. The square root ability of the previous circuit can be removed at a savings cost of four switches. One no longer needs the extra clock or the two switches that are used to create the generated one and shift the current quotient bit into the square root quotient delay line.

The previous device assumes each iteration is 2N cycles. This time is required for the square root operation. By eliminating the square root function, it is possible to implement a divider square rooter which has an iteration length of only N time units. Unfortunately, a most significant bit first multiplier cannot be designed with an iteration time of only N cycles since the carries must propagate along the entire 2N length of the final product. A least significant bit first multiplier must be used in any N cycle per iteration device. Such a multiplier does not map well onto a device used for division. The net result is that an N cycle per iteration divider/multiplier is rather expensive. Such a circuit can be implemented with 36 switches.

Table 1. Switch Counts for Division

Device	Switch Count
Restoring Divider Using Addition	20
Restoring Divider Using Copies	23
Non-Restoring Division	28
Square Rooter	23
Divider/Square Rooter	26
Multiplier/Divider/Square Rooter	31
Multiplier/Divider	27

13. Conclusions

This paper looked at several different division algorithms. Several algorithms were proven to be impractical to implement using the limited number of optical switches available. The algorithm most suited for optical implementation is the simple shift and compare algorithm. Both restoring and non-restoring shift and compare algorithms were designed and compared. The best divider consisted of a version of a restoring divider in which the restoration of the dividend was accomplished by simply keeping an unaltered copy.

Other possible designs were then considered. Parallel dividers which operated on two or more bits at a time were found to be seemingly impractical. Although an N bit at a time divider is possible, its cost in switch count makes it very impractical. Making a device that can do both square rooting and division, on the other hand, certainly seems reasonable. This is due to the similarity of both functions. Finally, a divider/square rooter was presented. This appears to be the simplest extension of a simple divider. Such a device only requires twenty-five switches. Furthermore, the divider/square rooter appears as though it can be upgraded to also include a multiplier, thus making a possible complete functional unit. Of course, the frequency of the square root operation may make such a device unnecessary.

Table 1. shows a list of several implementations of bit serial devices along with their costs in switch count. Each switch count includes a timing device capable of detecting when an operation is in progress. Such a feature may be eliminated, thus reducing each switch count by three. The cost of the Restoring Divider Using Copies and the Non- Restoring Divider are slightly misleading. Both of these devices perform a division in half the time of the other circuits.

References

[AEGP67] S. Anderson, J. Ealle, R. Goldschmitt, and D. Powers, "The IBM System/360 Model 91: Floating-point Execution Unit," *IBM Journal of Research and Development*, Vol. 11, Jan. 1967, pp. 34-53.

[Atki67] D.E. Atkins, "The Theory and Implementation of SRT Division," *Report No. 230*, Dept. of Computer Science, University of Illinois, June 1967.

[HEJP88] V.P. Heuring, H.F. Jordan, and J.P. Pratt, "A Bit Serial Architecture for Optical Computing," *OCS Technical Report 8801*, Dept. of Electrical and Comp. Eng. , University of Colorado, Boulder, Jan. 1988.

[Maje83] S. Majerski, "Square-Root Algorithms for High-Speed Digital Circuits," *IEEE Proceedings 6th Symposium on Computer Arithmetic*, June 1983, pp. 99-102.

[PlDi89] A. Pleszkun, and M. Dickman, "Design Alternatives for Optical Bit-Serial Multiplication," *OCS Technical Report 89-37*, Dept. of Electrical and Comp. Eng., University of Colorado, Boulder, Nov. 1989.

[Tayl85] George S. Taylor, "Radix 16 SRT Dividers with Overlapped Quotient Select Stages," *IEEE Proceedings 7th Symposium on Computer Arithmetic*, June 1985. pp. 64-71.

[TrEr77] K.S. Trivedi and M.D. Ercegovac, "On-Line Algorithms for Division and Multiplication," *IEEE Trans. Computers*, Vol. C-26, No. 7, July 1977, pp. 681-687.

[Wake81] John F. Wakerly, *Microcomputer Architecture and Programming*, John Wiley and Sons, 1981, pp. 998-101.

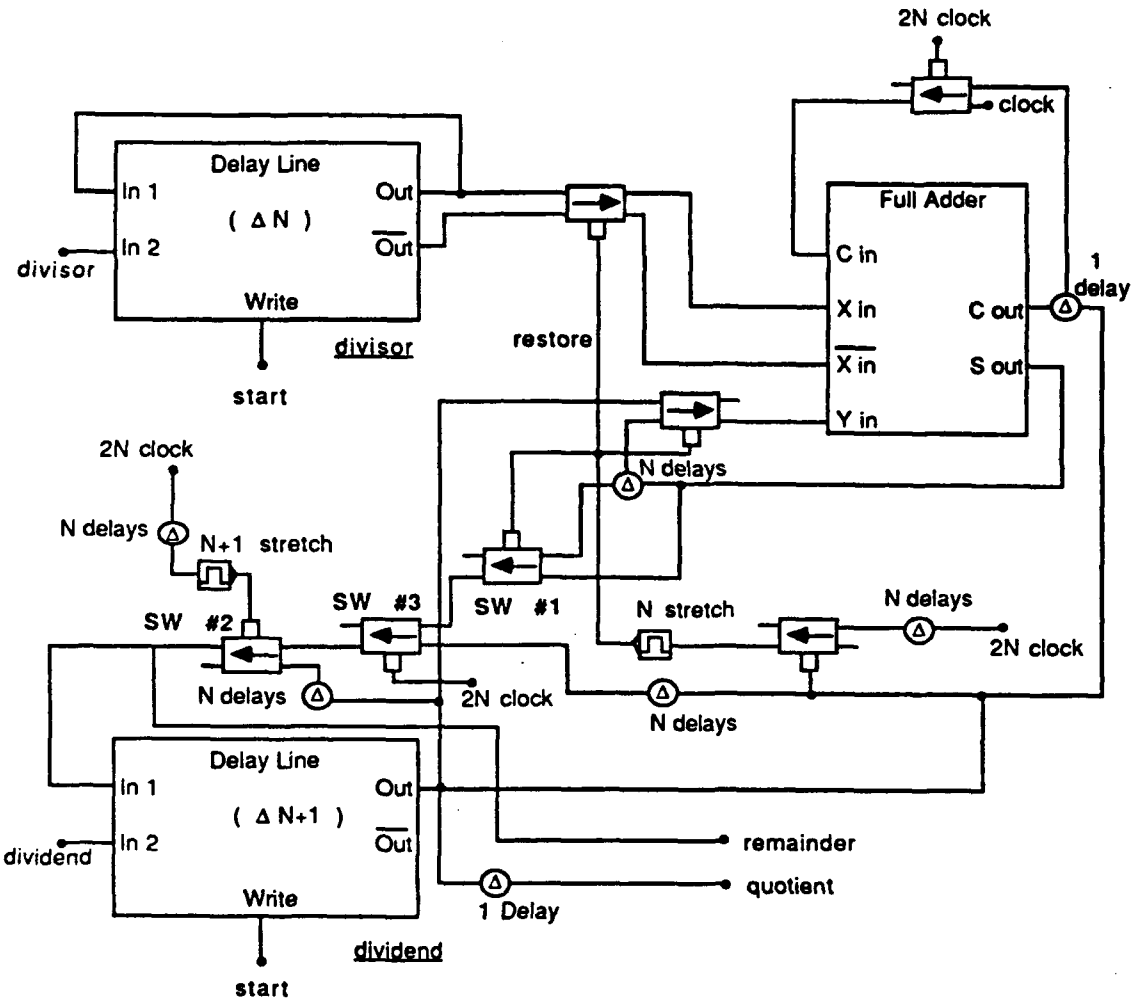


Figure 1. Restoring Divider Using Addition.

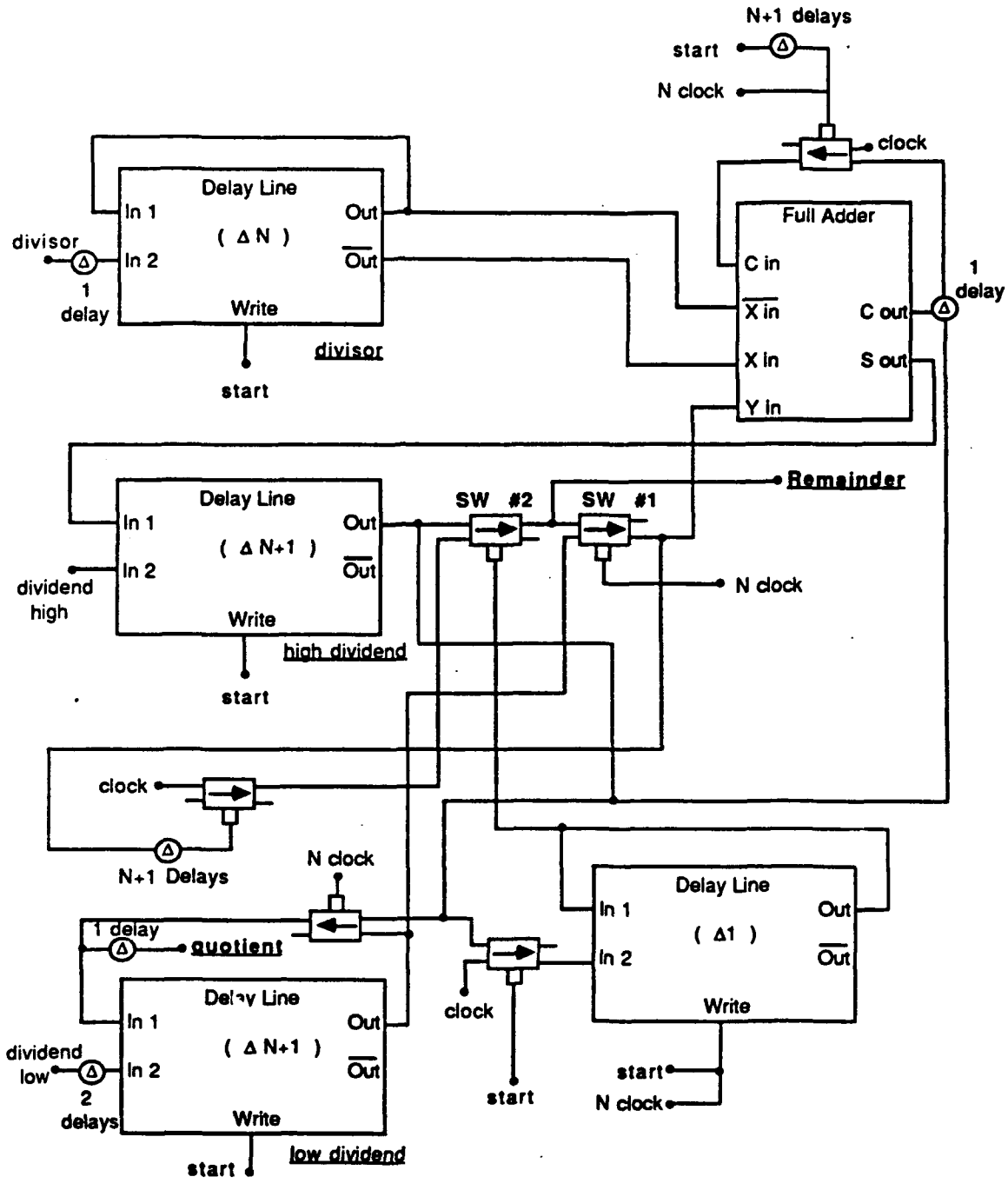


Figure 2. Restoring Divider Using Copies.

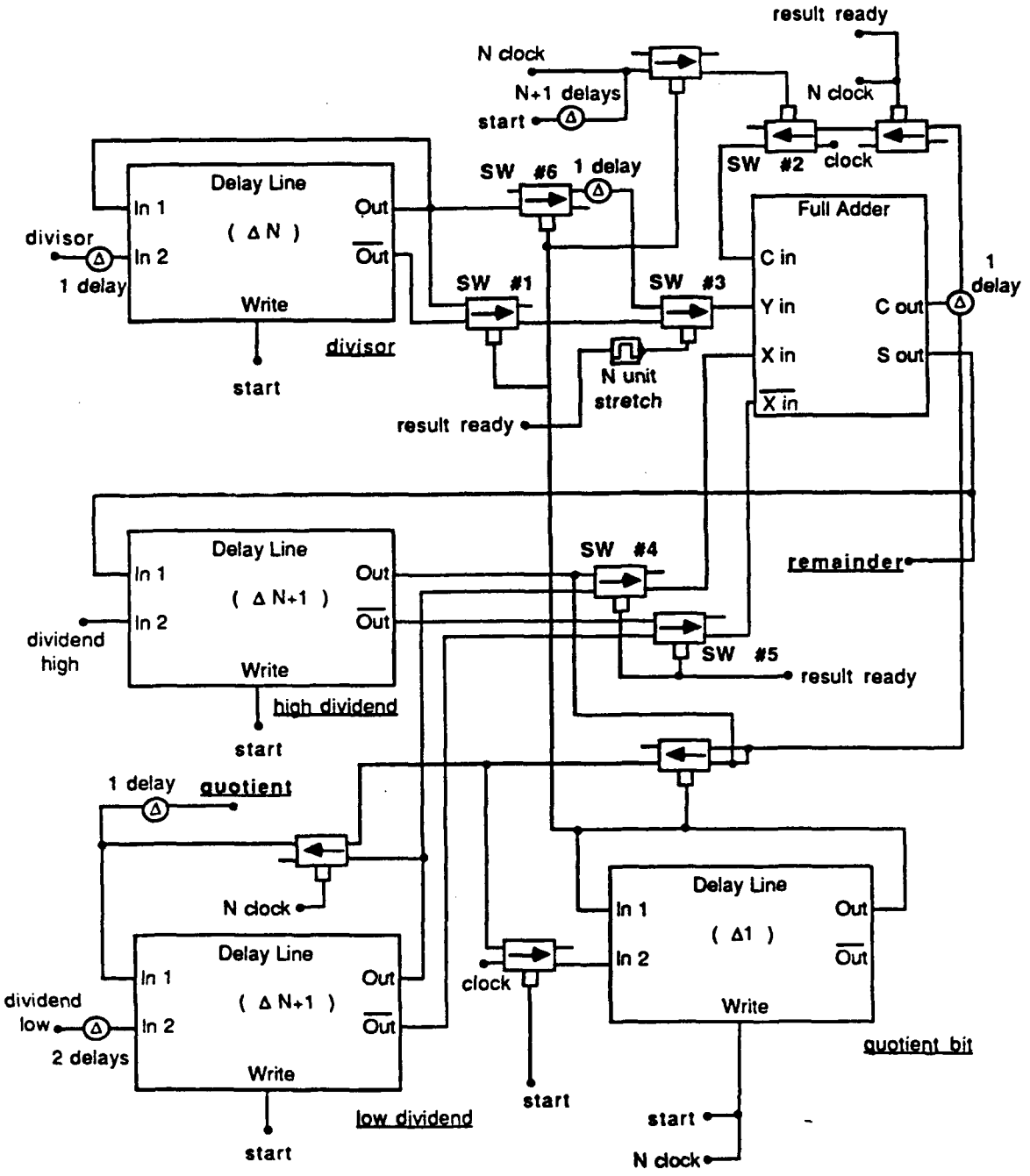


Figure 3. Non-Restoring Divider.

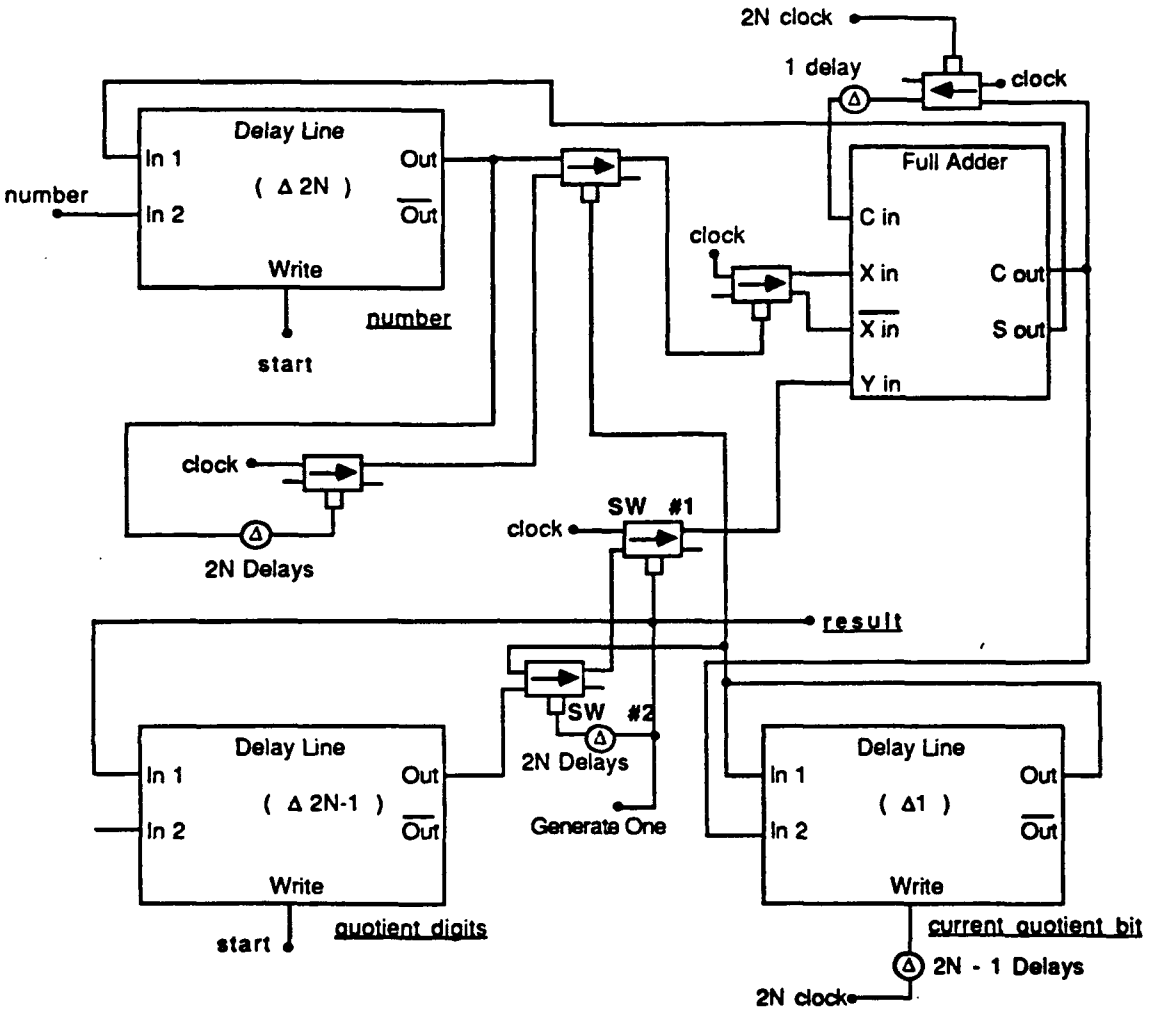


Figure 4. Square Rooter.

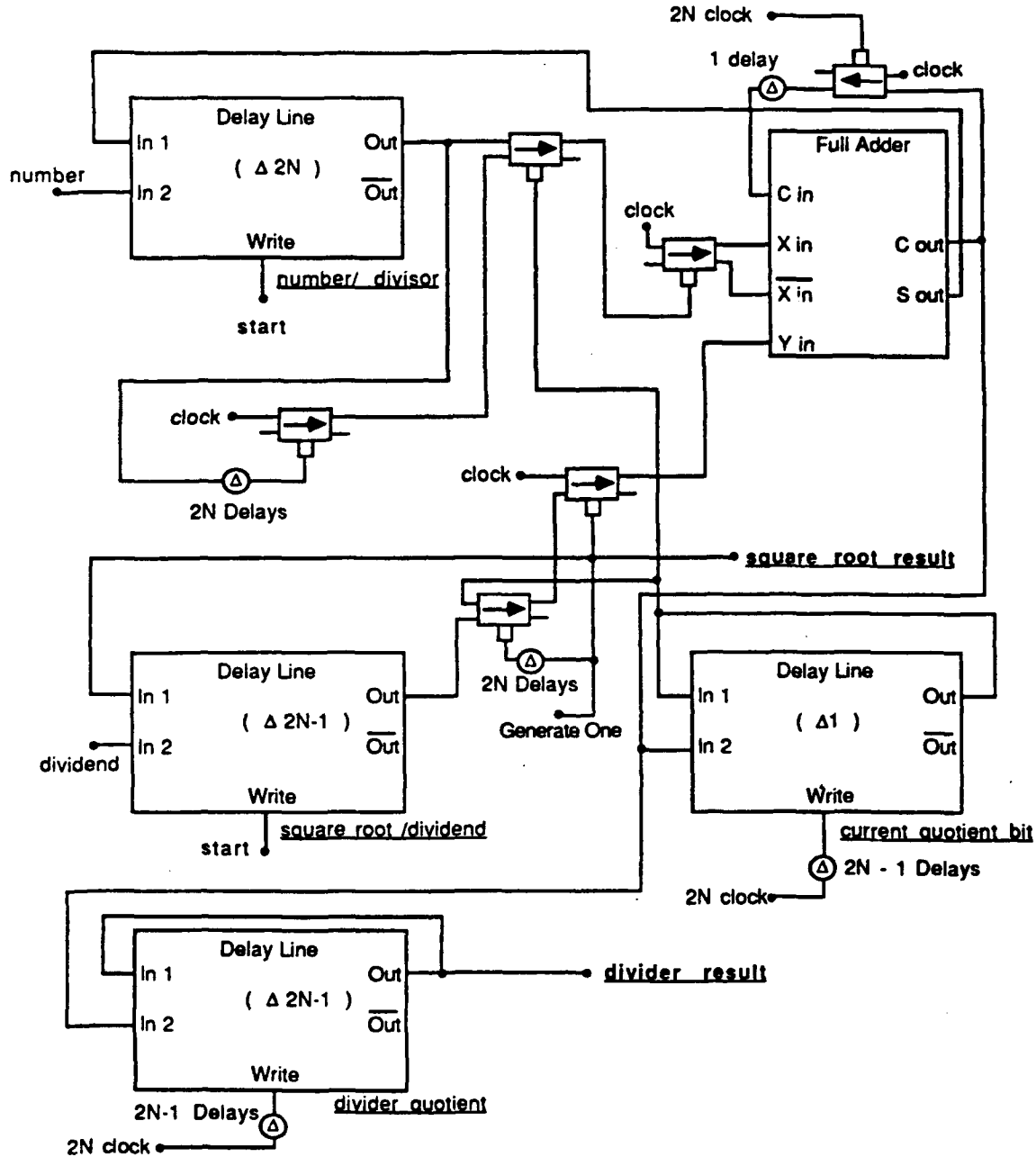


Figure 5. Divider/Square Rooter.

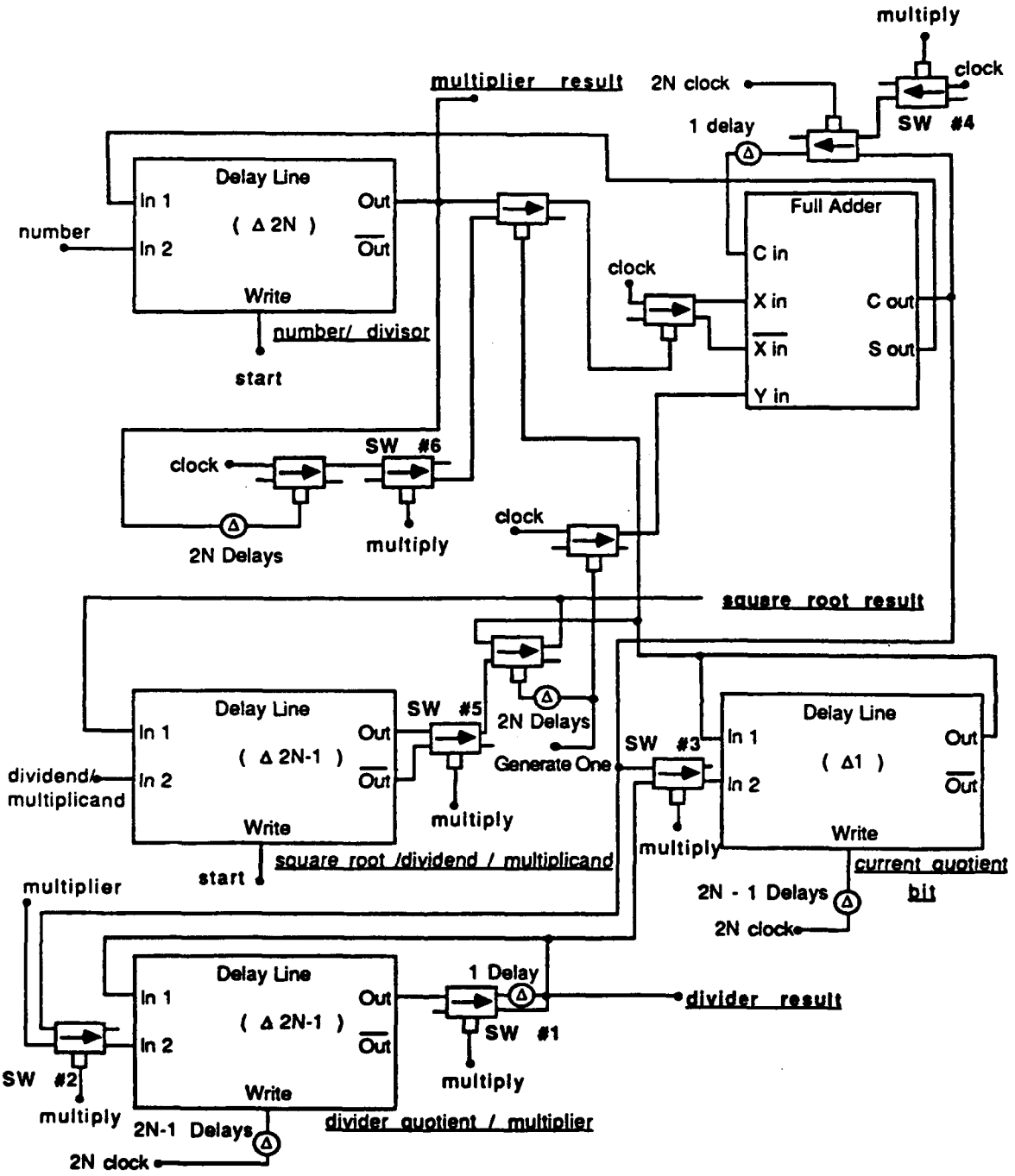
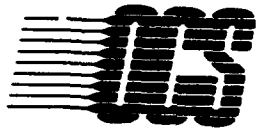


Figure 6. Multiplier/Divider/Square Rooter.

ATTACHMENT C

LITHIUM NIOBATE SWITCH EVALUATION

Robert Feuerstein



OPTOELECTRONIC COMPUTING SYSTEMS

AN

NSF

ENGINEERING RESEARCH CENTER

UNIVERSITY OF COLORADO
BOULDER, CO.

COLORADO STATE
UNIVERSITY
FORT COLLINS, CO.

LITHIUM NIOBATE SWITCH EVALUATION

Robert Feuerstein

Optoelectronic Computing Systems Center
Campus Box 525
University of Colorado
Boulder, CO 80309-0525

OCS Technical Report 90-16

The Center is supported in part by the NSF/ERC grant number CDR 8622236 and by the Colorado Advanced Technology Institute (CATI), an agency of the State of Colorado. CATI promotes advanced technology education and research at universities in Colorado for the purpose of economic development. This work was also supported in part by Rome Air Development Center (RADC) grant number Calspan C/UB-16.

LITHIUM NIOBATE SWITCH EVALUATION

Robert Feuerstein
Optoelectronic Computing Systems Center
Campus Box 525
University of Colorado
Boulder, CO 80309-0525

ABSTRACT

Lithium niobate ($LiNbO_3$) directional coupler switches may be connected with fiber optic cables, polarization controllers and passive splitters to form Boolean logic functions, e.g., NAND, NOR etc.; with which digital optical computers may be constructed. To design an operational computer, the characteristics of these components must be known. This report describes the experimental procedures that may be used to characterize both polarization-dependent, and polarization-independent $LiNbO_3$ switch performance. In addition, a procedure to measure the delay, or latency through the switches drive electronics is given.

LITHIUM NIOBATE SWITCH EVALUATION

Robert Feuerstein
Optoelectronic Computing Systems Center
University of Colorado at Boulder
Boulder, CO 80309-0525

1. INTRODUCTION

Lithium niobate ($LiNbO_3$) directional coupler switches may be connected with fiber optic cables, polarization controllers and passive splitters to form Boolean logic functions, e.g., NAND, NOR etc.; with which digital optical computers (DOC) may be constructed[1]. To design an operational DOC, the characteristics of these components must be known. Sarrazin[2] and Koehler and Bowers[3] describe the polarization controllers' properties used with the polarization-dependent switches. Characteristics of optical fibers important to their use in DOC's are discussed by Sarrazin[4]. This report describes the experimental procedures that may be used to characterize the $LiNbO_3$ switch performance.

Operationally, the $LiNbO_3$ electro-optically switched directional couplers are two optical-input, two optical-output devices with an optical control port denoted as Terminal C in which an optical pulse is converted into an electrical pulse that is then applied to the switching electrode. There are two waveguide configurations available: the Reversed $\Delta-\beta$ and the Balanced Bridge[5-7]. We are not concerned here with the actual internal arrangement of the switches, and the procedures given here work equally well for both configurations.

Part 2 of this memo defines the switch operation and the parameters of interest. Part 3 gives some general techniques for taking the measurements and some cautions to be observed. Part 4 outlines the measurement procedures for use with discrete optical components. Part 5 describes the use of the Opto-Electronics Inc. millimeter Resolution Optical Time Domain Reflectometer (OTDR) in switch evaluation. The OTDR may be used to examine the quality of the fiber-to-$LiNbO_3$ interfaces, as well as all other optical "joints" in the measurement setup. The OTDR can measure insertion loss, return loss and separation distances directly. Part 6 gives procedures for measuring the delay, or latency, in the Terminal C electronics. In a system with feedback loops, the latency places an upper limit on the clock frequency. In Part 7 we discuss interpretation of the switch parameters and how they affect switch performance.

2. $LiNbO_3$ SWITCH PARAMETERS

The switches may be in one of two states: the *bar state* or the *cross state*, as defined in Figure 1. The lines indicate the path taken by light entering the appropriate port. The switches may be characterized by the following operational parameters: 1) the optical insertion loss (L) from the input to the output port, in both states, 2) the crosstalk ratio (CR), 3) the extinction ratio (ER), 4) the contrast ratio (CON), and 5) the return loss (RL). The required bias voltage(s) and switching voltage for optimum switch performance must also be determined (there may be no bias voltage for polarization-independent switches such as those manufactured by Crystal Technology, Inc.). Note that these "optimum" settings will depend on how the switch is used. For example, with a switch using only one input port and one output port, the voltage settings for a minimum CR for port A may result in a large CR for port B. The following paragraphs define these parameters.

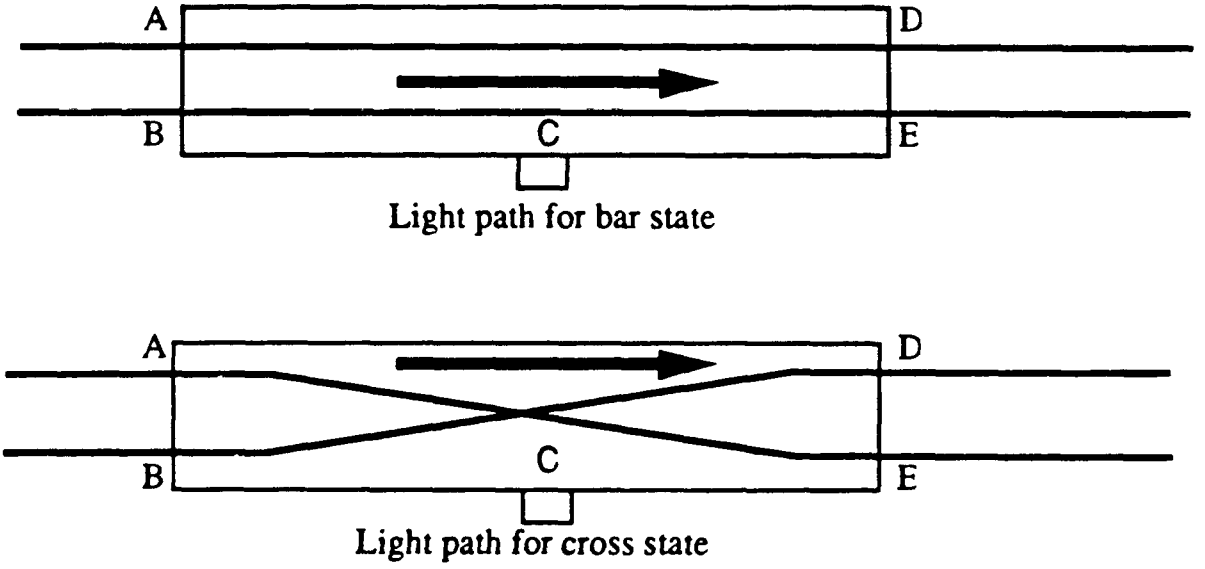


Figure 1. Diagram of two switch states, bar and cross. Terminal C is the switching (electrical) terminal. Terminals A and B are the optical inputs and terminals D and E are the optical outputs.

The crosstalk ratio in one switch state (first subscript) with light input at the indicated port (second subscript), $CR_{\text{switch state, input port}}$, with no light input at the other port, is defined as follows:

$$CR_{=,A} \equiv 10\ log(P_E/P_D) \quad (1)$$
$$CR_{x,A} \equiv 10\ log(P_D/P_E)$$
$$CR_{x,B} \equiv 10\ log(P_E/P_D)$$
$$CR_{=,B} \equiv 10\ log(P_D/P_E)$$

where P_D and P_E are the light output powers of the appropriate port. The CR must be minimized to maintain a large signal-to-noise ratio (SNR) in the DOC and reduce false triggering of any switch receiving the signal as an input to Terminal C. The switching voltages for minimum CR need to be determined for both inputs separately. They may differ for the different input ports.

The extinction ratio in one switch state (first subscript) with light input at the indicated port (second subscript), $ER_{\text{switch state, input port}}$, with no light input at the other port, is defined as follows:

$$ER_{=,A} \equiv 10\ log(P_E/P_A) \quad (2)$$
$$ER_{x,A} \equiv 10\ log(P_D/P_A)$$
$$ER_{=,B} \equiv 10\ log(P_D/P_B)$$
$$ER_{x,B} \equiv 10\ log(P_E/P_B)$$

where P_A and P_B are the light input powers.

The insertion loss in one switch state (first subscript) with light input at the indicated port (second subscript), $L_{\text{switch state, input port}}$, with no light input at the other port, is defined as follows :

$$L_{=,A} \equiv 10\ log(P_D/P_A) \quad (3)$$
$$L_{x,A} \equiv 10\ log(P_E/P_A)$$
$$L_{=,B} \equiv 10\ log(P_E/P_B)$$
$$L_{x,B} \equiv 10\ log(P_D/P_B)$$

These three parameters are not independent, as it may easily be seen that

$$L_{x,A} = ER_{x,A} - CR_{x,A} \quad (4)$$

and similarly for each state and input pair.

The contrast ratio at an output port (first subscript) due to light input at the indicated port (second subscript), $CON_{\text{output port, input port}}$, with no light input at the other port, is defined as follows:

$$CON_{D,A} \equiv 10\ log(P_{D,x}/P_{D,=})$$
$$CON_{E,A} \equiv 10\ log((P_{E,=}/P_{E,x})$$
$$CON_{D,B} \equiv 10\ log((P_{D,=}/P_{D,x})$$
$$CON_{E,B} \equiv 10\ log((P_{E,x}/P_{E,=}) \quad (5)$$

with $P_{D,x}$ being the output power in the cross state and $P_{D,=}$ being the output power in the bar state, for port D, and similarly for port E..

The return loss (RL) at an optical joint is 10 *log* R, where R is the power reflection coefficient at that joint. For example, the fiber-to-fiber connections where the switch is connected into the measurement setup and the fiber-$LiNbO_3$ butt joint inside the switch package each have a certain reflectivity which can be characterized by the RL. This can only be measured by the OTDR since it allows separation in space/time of the reflected signals.

The bias voltage (BV) should be adjusted for the smallest CR when the switch is in its natural state (i.e., switching voltage is zero). This is generally the cross state, but this should be verified for the switch being tested. Once this is done, the bias voltages should not be changed for the remainder of the tests. However, due to temperature effects and charges in the surface layer of the $LiNbO_3$, the bias voltages may change with time, so they should be rechecked periodically. The switching voltage (SV) is that voltage which results in the smallest CR when in the bar state since we are assuming that the cross state is the natural state.

3. GENERAL CONSIDERATIONS

When performing the optical measurements required for the switch characterization, one should use index matching liquid for all connections between the fiber optic components. Also, these components are temperature sensitive, so the measurements should be performed in a constant temperature environment. There is a difference between single mode (SM) and multimode (MM) couplers (bulkheads) and connectors. The switches are SM, so all the fibers, couplers, splitters, bulkheads etc. should be SM (except for the fiber pigtailed to the optical detector/meter, which can be MM). The devices and the optical power meters or pulse detectors used are wavelength sensitive. Measurements should be performed at the wavelength of switch operation.

These directions assume all components have fiber-optic connectors attached. Splices can be used in place of connectors; however, they must be mechanically stable and the transmitted power should not be affected by vibrations.

For the polarization-dependent switches, the polarization controller or "butterfly" should be securely fastened to the table and connected to the switch input port with the shortest possible length of fiber. Dangling fibers should be taped down so as to prevent changes due to vibrations or other movements. The light source used must be linearly polarized, since poorly polarized light results in a large CR.

Titanium-diffused $LiNbO_3$ switches have an upper limit on the incident optical intensity due to photorefractive effects. A typical safe operating level is 5 milliwatts or less incident optical power into a single mode waveguide.

4. MEASUREMENT PROCEDURES WITH DISCRETE COMPONENTS

To characterize the switches, all that is necessary is to measure the absolute power of the incident and the transmitted light. One optical detector is sufficient, though two are more convenient. If two detectors are used, they should be carefully calibrated against each other for a range of optical signal levels. Their linear range of operation should be checked by increasing the optical power until the response is no longer linear by comparison with an optical power meter. If one is not available, be certain to stay below the manufacturer's specified limits. The measurements may be taken with CW or pulsed light and with DC or pulsed switching voltage applied.

The suggested experimental setup for the polarization-independent switches is shown in Fig. 2. The laser diode is connected through a variable attenuator to one switch input. The power is set low enough for linear detection and high enough to obtain a reasonable signal level. If there is a bias voltage electrode, it should be tuned for the minimum CR in the cross state as the first step. Tune it for both input ports and note both voltage settings if they are different. Then apply a switching voltage to Terminal C and note the voltage for which the smallest CR is obtained. Repeat this procedure for light input at the other input port.

The suggested experimental setup for the polarization-dependent switches is shown in Fig. 3. The laser light passes through a polarizer, a variable attenuator, the polarization controller (butterfly) for polarization rotation, and then into the switch. For ordinary switch fiber, strip any outer jacketing off the fiber pigtailed to the switch for a length sufficient to wind it inside the butterfly. If the fiber pigtailed to the switch is polarization maintaining (PM) fiber, a separate piece of non-PM fiber should be stripped for a length sufficient to wind it inside the butterfly, which should then be coupled to the switch PM fiber. The butterfly rotates the polarization so as to align it with the preferred orientation for

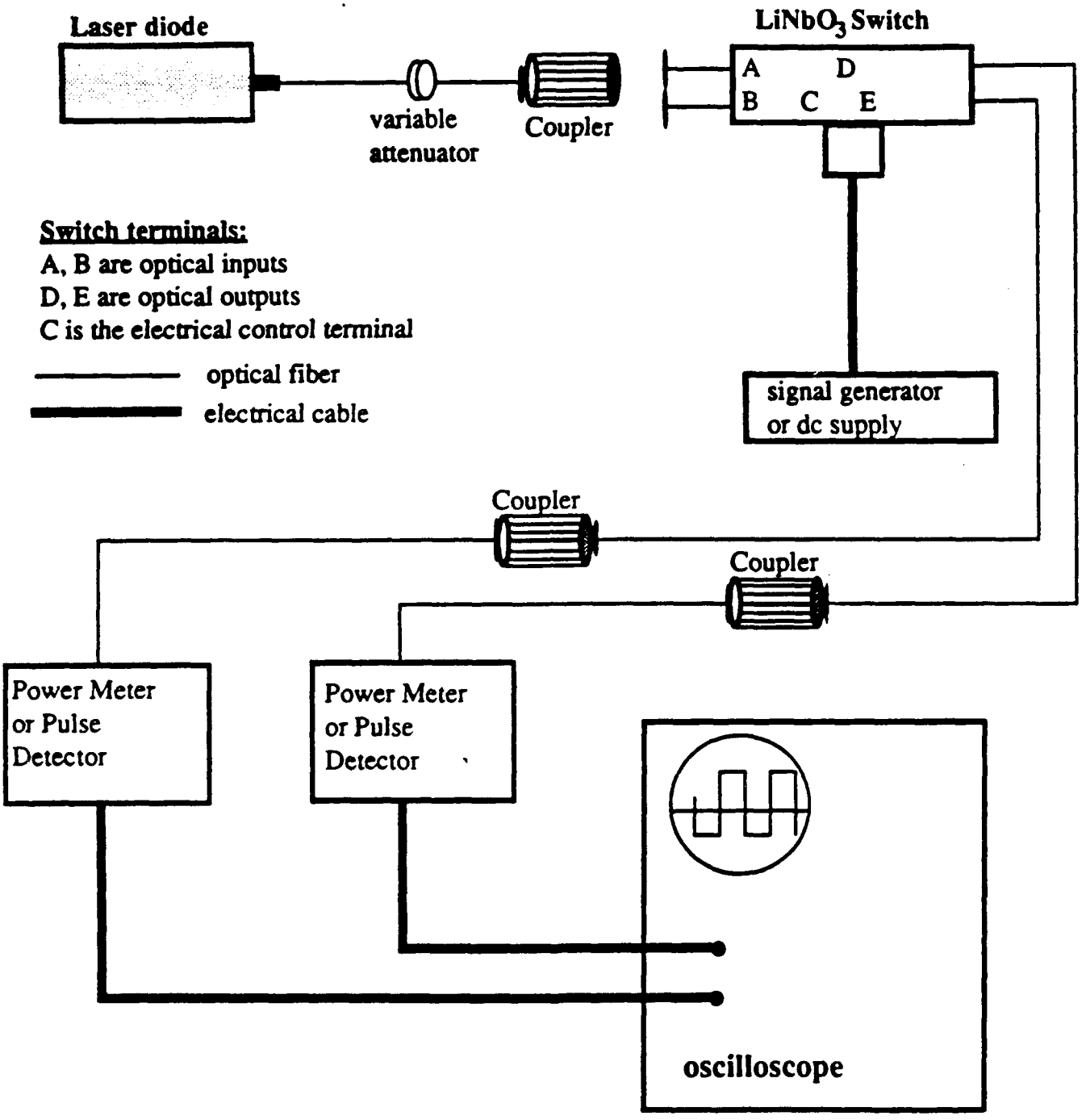


Figure 2. Experiment for characterizing the polarization-independent switches.

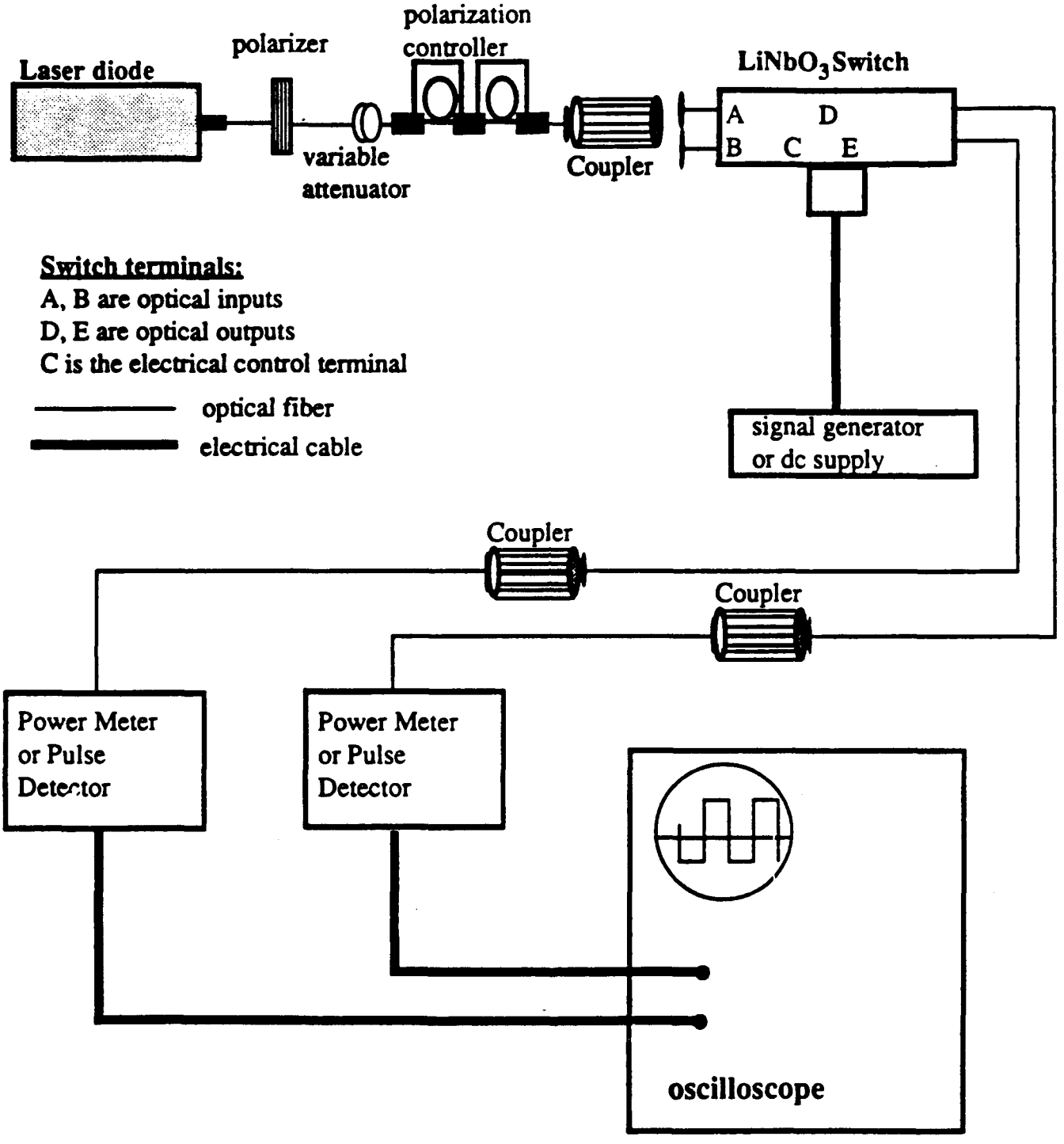


Figure 3. Experiment for characterizing the polarization-dependent switches.

the switch or the PM fiber. The butterfly should be adjusted for the minimum CR in the cross state. There are multiple positions of the butterfly for this to occur and some of these positions are better than others. If there is a bias voltage electrode, then this should also be tuned for the minimum CR in the cross state. Tune the BV for both input ports and note both voltage settings if they are different. One should obtain at least the minimum CR specified by the manufacturer. If the CR does not meet the manufacturer's specifications, adjust both the butterfly and the bias voltage setting until it does. If after repeated attempts the CR is still out of spec, reject the switch.

To characterize the switches, the only measurements necessary are the output powers for the two different inputs in both switch states. Once these are obtained, the CR, ER, CON and insertion loss can be calculated. A sample data sheet is given in Figure 4.

5. OTDR MEASUREMENTS

With the Opto-Electronics Inc. OTDR, loss measurements are simplified. Simply by connecting the switch through a short length of fiber to the OTDR, as shown in Fig. 5, and pressing a few buttons the return loss and the insertion loss are read out directly in dB. This instrument can also show the quality of all fiber optic interconnections. The magnitude of the reflected signal at any point in the measurement setup can be seen directly on the oscilloscope, which can be compared to a good connection. These readings can be saved and tabulated for later reference. See the OTDR Operating Manual for detailed instructions on its use. Application Note #10 from Opto-Electronics Inc. gives a detailed discussion of loss measurements using the OTDR.

6. MEASUREMENT OF TERMINAL C DELAY (LATENCY)

The measurement of the switch drive electronics delay, or latency, is easily accomplished with the setup in Fig. 6. It is necessary to know the lengths of the fibers in all the connections and the fiber index of refraction, n, in the core. With the index, it is possible to calculate the time delay t in ns of fibers of length l in cm using the expression

$$t = n\, l/30. \qquad (6)$$

The error introduced by neglecting the different index in the short section of $LiNbO_3$ is negligible.

Date:	**Wavelength:**
Switch Number:	**Temperature:**
Polarization Dependent?:	**Bias Voltage:**

INPUT		OUTPUT		Switching Voltage	Loss	CR	ER	CON
Port	Power in μW	Port	Power in μW					
A		D						
A		E						
A		D						
A		E						
B		D						
B		E						
B		D						
B		E						

Fig. 4. Sample Data Sheet

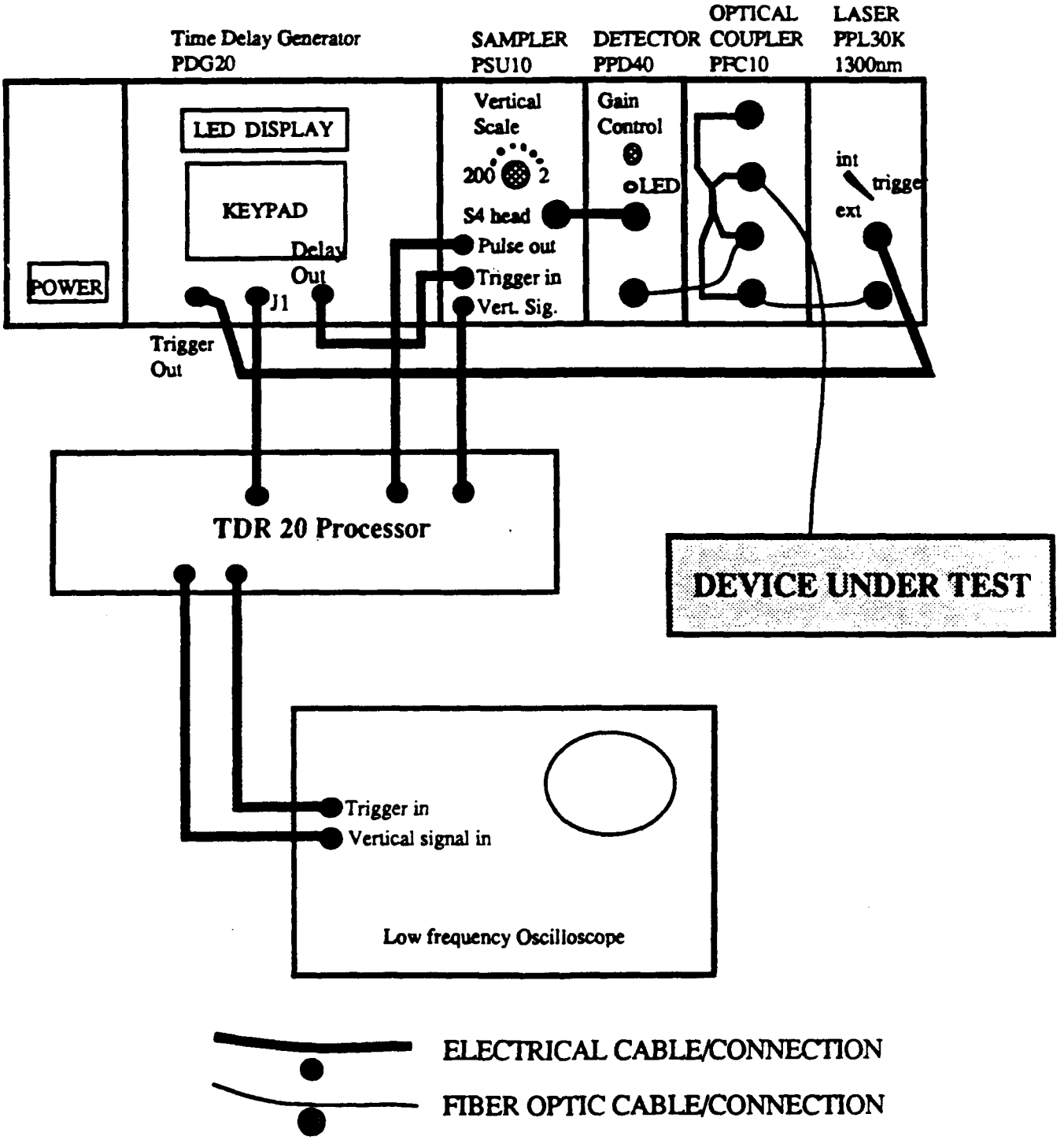


Fig. 5. OTDR setup for insertion loss and return loss measurements.

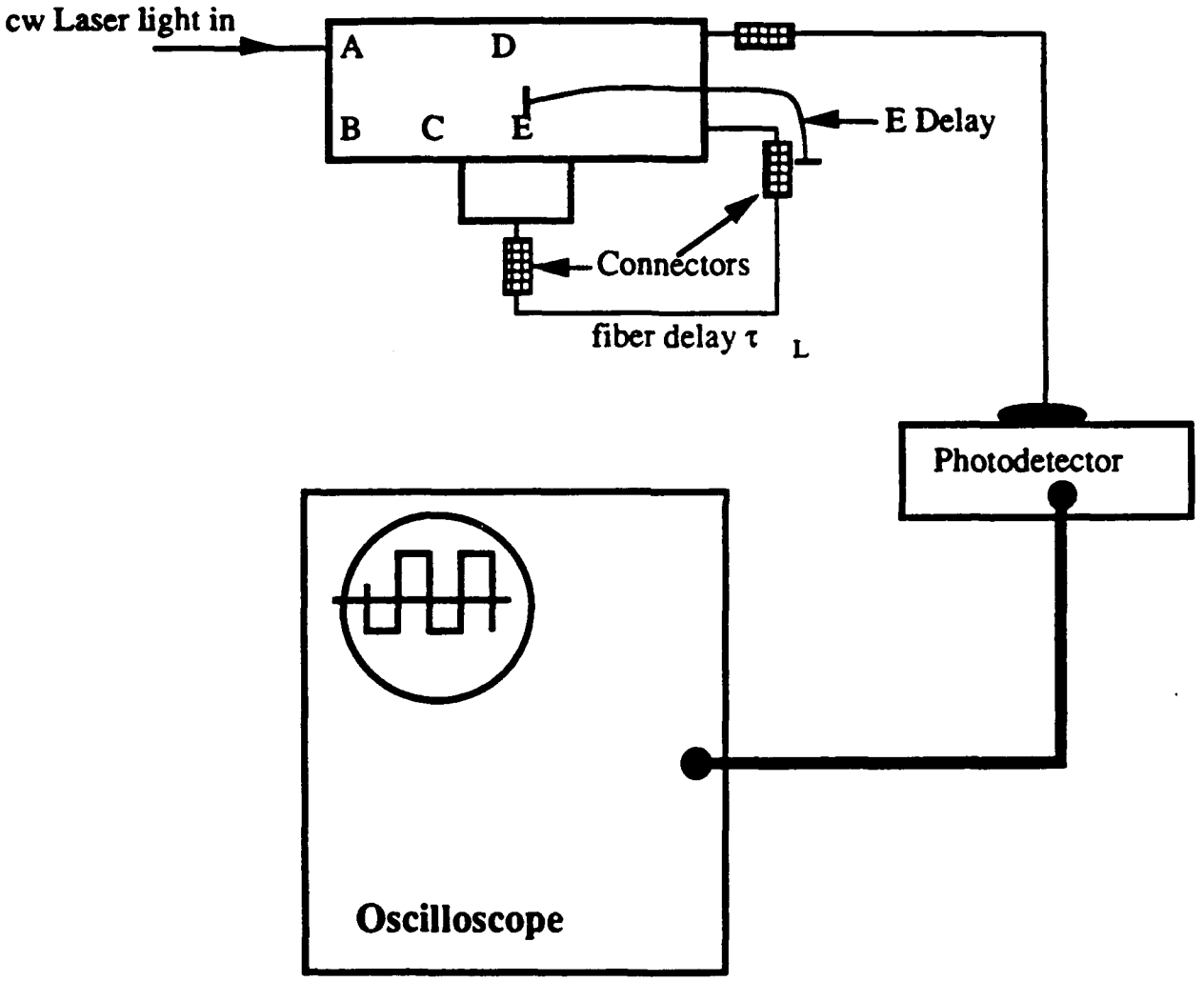


Fig. 6. Optical oscillator circuit for determining the delay in Terminal C electronics.

$$T = 2(E + \tau_L + \text{Latency}), \quad (7)$$

where E is the length from the $LiNbO_3$ switch center to the port E fiber end in ns as shown in Fig. 6. Thus by measuring the period of the oscillation on the oscilloscope, one can determine the delay in Terminal C, inclusive of any fiber to the Terminal C detector, the electronics, and any electrical cable to the switch electrode. One must be sure that there is no stretching of the pulses by the electronics, otherwise the measured latency time will be larger than the actual latency time as shown in Fig. 7. Eq. 7 is now changed to

$$T = 2(E + \tau_L + \text{Latency}) + \sigma \quad (8)$$

where σ is the stretch introduced by the electronics at the end of the light pulse.

This circuit setup in Fig. 6 is an oscillator. With continuous-wave laser light entering port A, light comes out port E, since the switch is initially in the cross state. The light travels around the loop with a delay τ_L, after which it enters the Terminal C electronics. After the latency period, the switch changes states to the bar state and the light now exits port D. The switch stays in the bar state for the duration of the light pulse that was in transit, in the loop from E to C, plus the latency time. After the light "off" state is sensed by Terminal C, the switch returns to the cross state and the light again enters the feedback loop, after which the cycle repeats. Thus, this is a square wave light output generator whose period is twice the loop delay, including the latency. We have that the period of oscillation T is given by

7. EVALUATION OF SWITCH DATA

The most important operational parameters for the switch are the worst case CR, worst case insertion loss and the switching voltage. The CR determines what the SNR is. The larger the SNR, the more reliable the triggering of the switch. The tolerable level of CR depends on the sensitivity of the drive electronics and what the relative light power levels of different signals entering the same Terminal C electronics are. This will have to be empirically determined during testing, but a CR of -10dB should be adequate to ensure correct triggering of the switch.

The maximum tolerable insertion loss depends on the placement of the switch in the system. If a clock signal is input and the output of the switch only goes to a Terminal C input, 15 dB is tolerable. For a series

connection of three switches with a splitter or two, 5 dB maximum per switch would be required. Therefore, switches with losses larger than that specified by the manufacturer can be used in a DOC, if they are carefully used.

The drive electronics for the system has a maximum voltage swing which cannot be exceeded. If the switching voltage exceeds this value, then the switch is unusable with the drive electronics as designed.

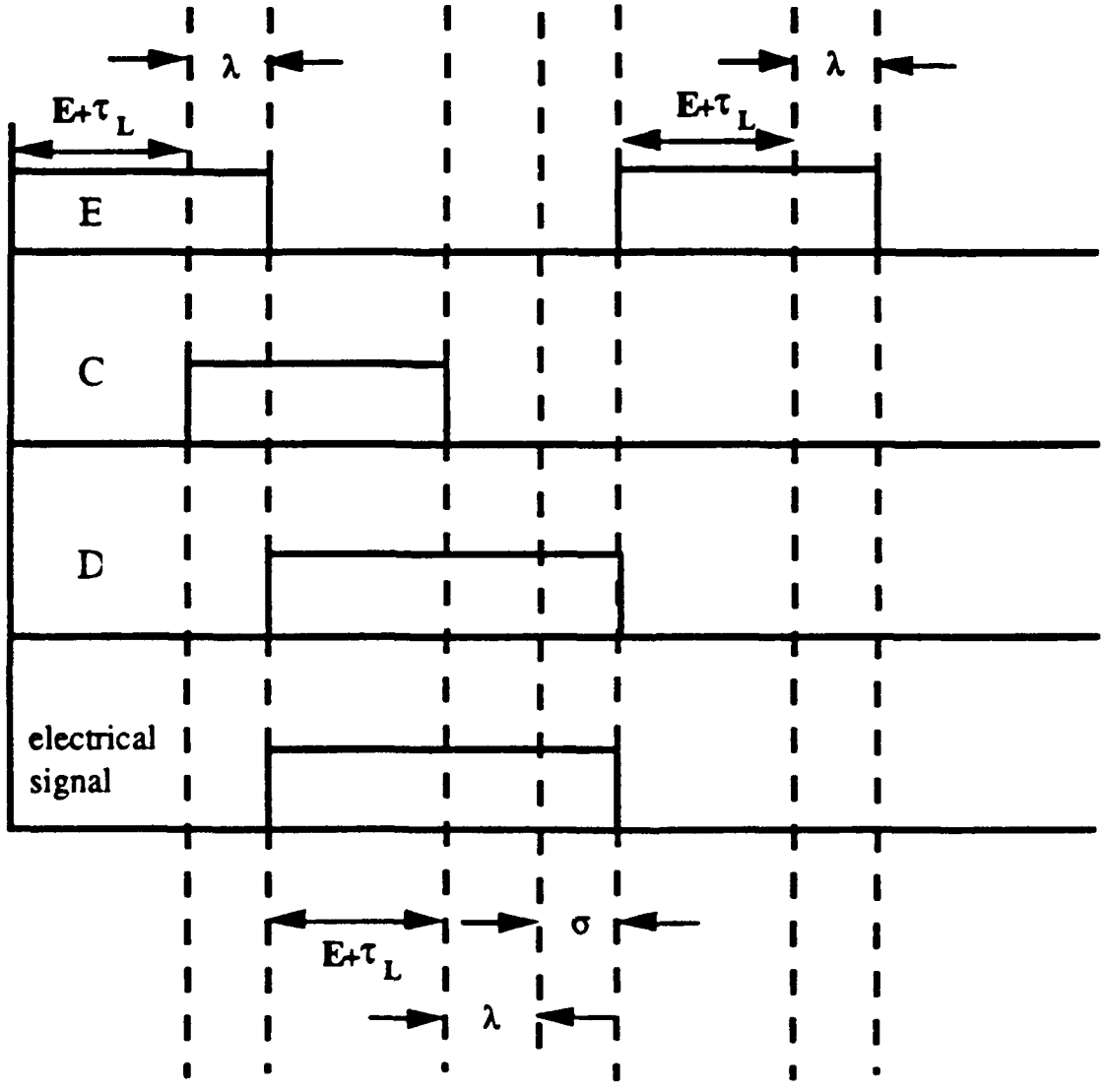


Fig. 7. Depiction of how stretch at the end of the pulse changes measurement of the latency. σ is the stretch, λ is the latency, and $E+\tau_L$ are the terminal E and loop delay.

8. REFERENCES

[1]H. F. Jordan , *Fiber Optic Computer Architectures*, OCS Technical Report 90-11, Optoelectronic Computing Systems Center, Univ. of Colorado, Boulder, Jan 1990.

[2]D. B. Sarrazin, *Behavior of Miniaturized Polarization Controllers*, OCS Technical Report 89-49, Optoelectronic Computing Systems Center, Univ. of Colorado, Boulder, Dec 1989.

[3]B. G. Koehler and J. E. Bowers, "In-Line Single-Mode Fiber Polarization Controllers at 1.55, 1.30 and 0.63 μm", Appl. Opt. 24(3), 349, 1 Feb 1985.

[4]D. B. Sarrazin, H. F. Jordan and V. P. Heuring, *Fiber Optic Delay Line Memory*, Appl. Opt., 29(5), 627-637, 10 Feb. 1990.

[5]A. Yariv and P. Yeh, *Optical Waves in Crystals: Propagation and Control of Laser Radiation*, John Wiley and Sons, NY, 1984.

[6]S. Wang, "Principles and Characteristics of Integrable Active and Passive Optical Devices", in *Semiconductors and Semimetals*, Vol 22, Part E, W. T. Tsang Ed., Academic Press 1985.

[7]K. Iizuka, *Engineering Optics*, Springer-Verlag, 1987.

MISSION

OF

ROME LABORATORY

Rome Laboratory plans and executes an interdisciplinary program in research, development, test, and technology transition in support of Air Force Command, Control, Communications and Intelligence (C^3I) activities for all Air Force platforms. It also executes selected acquisition programs in several areas of expertise. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C^3I systems. In addition, Rome Laboratory's technology supports other AFSC Product Divisions, the Air Force user community, and other DOD and non-DOD agencies. Rome Laboratory maintains technical competence and research programs in areas including, but not limited to, communications, command and control, battle management, intelligence information processing, computational sciences and software producibility, wide area surveillance/sensors, signal processing, solid state sciences, photonics, electromagnetic technology, superconductivity, and electronic reliability/maintainability and testability.

END →